UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INDUSTRIAL

PROYECTO FIN DE CARRERA

ENTORNO DE SIMULACIÓN BASADO EN MARILOU Y MATLAB PARA CONTROL ANTIBALANCEO DE PUENTES GRÚA.

AUTOR: **SILVIA GONZÁLEZ ÁLVAREZ**

TUTOR: **Mª DOLORES BLANCO ROJAS**

LEGANÉS, JULIO DE 2012

A mis padres, por el apoyo que siempre me dan.

Título: **Entorno de Simulación basado en Marilou y Matlab para Control Antibalanceo de Puentes Grúa.**
Autor: **Silvia González Álvarez**
Directora: **Mª Dolores Blanco Rojas**

EL TRIBUNAL

Presidente: María Malfaz

Vocal: José Luis González

Secretario: César Arismendi

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 17 de Julio de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

A mi tutora, Mª Dolores Blanco Rojas, por su paciencia, apoyo y dedicación a lo largo del desarrollo de este proyecto con el que culmino mi formación como ingeniera.

A los compañeros con los que compartí laboratorio durante el desarrollo del proyecto, por hacer más interesante y divertido el trabajo y por ayudarme con las dudas repentinas.

Al Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid, por brindarme las instalaciones para poder desarrollar este proyecto.

A todos aquellos profesores que han contribuido en mi formación y han hecho que pueda cumplir el sueño de ser Ingeniera Industrial.

A mis padres, Ángel y Rosma, por su perseverancia y apoyo incondicional; sin ellos no hubiera llegado hasta aquí. Por los días en Madrid en los que no hicieron otra cosa que leer y cocinar para que yo pudiera dedicarme a mis estudios.

A mi novio y amigo José Felipe, por apoyarme y animarme a seguir cuando el optimismo disminuía. Por acompañarme a lo largo de esta gran aventura y seguirme a un nuevo mundo.

A mi familia, porque siempre están cuando los necesitas.

Y por último, pero no menos importante, a todos aquellos compañeros y amigos con los que compartí horas de biblioteca, prácticas, exámenes y cañas.

Gracias a todos por el tiempo compartido.

Silvia González Álvarez

RESUMEN

Dentro del marco del proyecto TechnoFusión se plantea el estudio de sistemas de control remoto, como el desarrollo de puentes grúas de gran tonelaje que incluyan técnicas de posicionamiento preciso y de sistemas antibalanceo. En este sentido, se ha desarrollado en este proyecto un modelo de una grúa puente sobre el que probar algoritmos de control antibalanceo.

Se ha elegido la herramienta Marilou para desarrollar el modelo y la simulación de la grúa puente. Esta herramienta permite la construcción del modelo de la grúa a través de formas sencillas y dispone, además, de sensores y actuadores predefinidos que se pueden utilizar para llevar a cabo el control de la grúa y la obtención de medidas de oscilación.

Para evaluar la herramienta de simulación se ha elegido un algoritmo de control antibalanceo basado en *"Input Shaping"* que se desarrolla, a modo de prueba, en sus modos más básicos. Este algoritmo se ha desarrollado con funciones programadas en Matlab, donde también se ha diseñado una interfaz gráfica para el control de la grúa y la visualización de las oscilaciones que se obtienen como resultado.

Palabras Clave: Marilou, Input Shaping, Antibalanceo, TechnoFusión, puentes grúas.

ABSTRACT

Within the framework of the TechnoFusión project it is considered the study of remote control systems such as the development of large overhead cranes including anti-swinging systems and precise positioning techniques. In this regard, it has been developed in this project a model of overhead crane on which anti-swinging algorithms can be tested.

For the development of the model and the simulation of the overhead crane it was chosen the tool Marilou. This tool allows the construction of the crane model through simple shapes and it also has predefined sensors and actuators that can be used to carry out the control of the crane and for the obtention of oscillation measurements.

To evaluate the simulation tool, it has been chosen an anti-swinging control algorithm based on Input Shaping which has been developed, in a trial version, with the most basic functions. This algorithm has been developed with functions programmed in Matlab, where it has also been designed a graphical interface for the control of the crane and the visualization of the resulting oscillations.

Key words: Marilou, Input Shaping, Anti-Swinging, TechnoFusion, Overhead crane.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1. TechnoFusión

La fusión nuclear se presenta como una de las pocas opciones energéticas inagotables y respetuosas con el medio ambiente capaz de suministrar energía a escala suficiente como para cubrir la demanda predecible para las próximas décadas. En comparación con la fisión, la fusión es casi inocua ya que su combustible (isótopos de hidrógeno) se obtiene casi de forma ilimitada del mar, es barato, no radiactivo y se encuentra distribuido geográficamente de manera uniforme. Por otro lado, posee un potencial energético más amplio que la fisión y no es una reacción en cadena, por lo que puede detenerse de forma instantánea cerrando sencillamente el suministro de combustible. En cuanto a los residuos, estos son de baja activación radiactiva (dejan de ser peligrosos en menos de 100 años desde que la central deja de funcionar y además no salen del corazón del reactor). Considerando además que no produce contaminación atmosférica que provoque lluvia ácida o el efecto invernadero, la fusión podría ser una de las maneras más ecológicas de solventar la acusada dependencia actual de los combustibles fósiles.

En el camino hacia los reactores de fusión, en los próximos 20-30 años se va a producir un gran desarrollo de la investigación en fusión y de sus tecnologías asociadas, desplazándose el esfuerzo principal de la investigación básica en física de plasma hacia el desarrollo tecnológico de los diferentes sistemas que formarán parte de los reactores de fusión, y que van a requerir el diseño y la construcción de grandes proyectos multinacionales.

En el ámbito anterior se enmarca el proyecto de construcción, en la Comunidad de Madrid, del Centro Nacional de Tecnologías para la Fusión (TechnoFusión) [1], una de las infraestructuras incluidas en el Mapa Nacional de Infraestructuras Científico-Técnicas Singulares (ICTS), y proyecto conjunto entre el Ministerio de Ciencia e Innovación y el Gobierno regional de Madrid. Esta ICTS se basa en la experiencia técnica del Laboratorio Nacional de Fusión del CIEMAT y de la UPM así como de otros Centros y Universidades de Madrid participantes como UC3M, UNED, UAM o CSIC.

El proyecto prevé la creación de las infraestructuras necesarias para el desarrollo de las tecnologías requeridas para los futuros reactores comerciales de fusión, asegurando la participación de empresas y de grupos de investigación españoles. Las condiciones que deberán soportar los componentes del reactor y las propiedades que de ellos se esperan los sitúan en un terreno desconocido que precisamente TechnoFusión pretende explorar. Por ello se propone la construcción de aquellas instalaciones necesarias para la fabricación, prueba y análisis de los materiales más críticos, así como para impulsar el desarrollo de simulaciones numéricas para el estudio del comportamiento de dichos materiales bajo condiciones tan exigentes y la investigación en equipos de manipulación remota para el necesario mantenimiento que conlleva el reactor.

Otro de los objetivos de TechnoFusión es situar a España entre los países en cabeza en la fase de viabilidad tecnológica de la Fusión, creando una infraestructura que favorezca la participación en el Programa Europeo de Fusión de las Universidades, Centros I+D y Empresas españolas involucradas en el proyecto.

Aunque la ICTS se centrará en las tecnologías de fusión, estas tecnologías también serán de interés para muchas otras actividades científicas y tecnológicas (fisión, ciencias biológicas, investigación espacial, aplicaciones médicas, producción de isótopos,...).

TechnoFusión, estará constituida como Instalación Singular con capacidad para desarrollar siete de las principales áreas de investigación relacionadas con las tecnologías de fusión: la producción y procesado de materiales, la irradiación de materiales, el estudio de la interacción plasma-pared, las tecnologías de metales líquidos, las técnicas de caracterización de los materiales, las tecnologías de manipulación remota y la simulación computacional. Muchas de estas áreas tecnológicas serán únicas en el mundo. TechnoFusión pretende agrupar recursos humanos y materiales suficientes con el objetivo de contribuir al desarrollo de una fuente segura, limpia e inagotable de energía para las generaciones venideras.

De entre estas áreas, tiene especial interés para este documento el área de investigación en tecnologías de manipulación remota. Entre las razones para recurrir a sistemas remotos en la realización de muchas de las operaciones dentro de las instalaciones del reactor se encuentran las grandes dimensiones y el peso de las piezas y herramientas a manejar y, sobretodo, la radiación presente por la activación de los equipos y materiales cercanos al dispositivo. Debido a esto, las plantas de energía de fusión futuras poseerán un mantenimiento constante y complejo por medio de manipulación remota, suponiendo estos sistemas un factor decisivo para la proliferación y viabilidad de los futuros reactores de fusión.

El objetivo básico inmediato del área científico-técnica de manipulación remota de TechnoFusión es crear instalaciones en las que probar y poner a punto los equipos y procedimientos de trabajo de telerrobótica que puedan contribuir al mantenimiento y reparación de instalaciones de fusión nuclear, principalmente ITER, DEMO e IFMIF. Además, esta área debe servir para promover el desarrollo de nuevas técnicas robóticas, compatibles con las condiciones en el interior de un reactor de fusión. Éstas serán incompatibles con la reparación o sustitución de sus componentes manualmente, siendo imprescindible su manejo por manipulación remota. Así, es de máxima importancia no solo el desarrollo de nuevas técnicas robóticas compatibles con estas condiciones hostiles, sino también la acreditación de las existentes para su uso en instalaciones como ITER o IFMIF. El tamaño de los componentes que se van a usar y las dificultades de su disposición en el espacio que se dispone provoca la necesidad de desarrollos hasta ahora no considerados en las técnicas de manipulación.

El área de manipulación remota de TechnoFusión deberá disponer desde el inicio de un mínimo de sistemas de manipulación generales para poder realizar tareas de validación. Además será necesario un equipamiento mínimo para competir internacionalmente con otros laboratorios que ya disponen de algún tipo de instalaciones robotizadas de ensayo y validación para fusión.

Tras estudiar las instalaciones disponibles en otros laboratorios y haciendo una estimación de las necesidades para actividades futuras, el área de manipulación remota de TechnoFusión ha definido como punto de partida unas instalaciones mínimas que se consideran necesarias para cubrir los objetivos de esta área de experimentación y ser competitivos internacionalmente.

Entre estas instalaciones, es necesario disponer de una grúa puente de grandes dimensiones, alta capacidad de carga y mucha precisión de posicionamiento. Esta grúa tendrá un papel imprescindible en el transporte de componentes de los diferentes sistemas o de los mismos robots que van a realizar labores de manipulación.

Debido a la especial sensibilidad del ambiente de trabajo estas grúas deberán disponer de sistemas antibalanceo y técnicas de posicionamiento preciso.

En relación a esta necesidad, tiene especial importancia la experiencia del grupo Robotics Lab de la Universidad Carlos III de Madrid en sistemas antibalanceo para el control de grúas [2].

De esta necesidad concreta dentro del proyecto TechnoFusión surge el desarrollo del proyecto que en este documento se expone, ya que trata de encontrar una herramienta adecuada para el modelado y simulación de grúas puente como la requerida en el área de trabajo de manipulación remota de TechnoFusión [3], sobre la que se pueda comprobar la eficacia de diferentes algoritmos de control antibalanceo.

1.2. Objetivos del proyecto

Cuando se desarrollan proyectos a gran escala o en ambientes especialmente complicados es importante poder modelar y simular el sistema en un ordenador antes de llevarlo a una escala real. Así se puede encontrar errores o incompatibilidades que, una vez llevado a cabo el proyecto en la realidad, difícilmente podrían resolverse.

Para que este estudio previo sea relevante, el programa elegido debe permitir crear un modelo funcional y estructuralmente correcto, es decir, debe estar construido de manera que los resultados obtenidos y las predicciones del simulador sean relevantes para la tarea que se intenta llevar a cabo.

En este caso, se ha considerado que la herramienta Marilou podría ser una buena opción para modelar la grúa de gran tonelaje prevista en TechnoFusión y para probar los algoritmos antibalanceo estudiados y desarrollados por el departamento Robotics Lab de la Universidad Carlos III de Madrid [2]. Pero antes que nada, se debe comprobar si efectivamente es la herramienta idónea. Ese es el objetivo principal de este proyecto, evaluar la idoneidad de Marilou para el modelado y simulación de un control antibalanceo sobre una grúa puente.

Marilou es un programa dirigido específicamente al diseño de robots, en el que se puede definir desde la forma del sistema a modelar hasta los sensores y actuadores que debe tener, todo en un entorno gráfico que hace que la tarea resulte más sencilla e intuitiva. Además, permite simular el modelo realizado en tiempo real y aplicar los algoritmos deseados en una simulación 3D en tiempo real.

Para la consecución del proyecto se utilizará, además del propio Marilou, que debe ser evaluado, el Microsoft Visual C++ para la programación en C++ y el programa Matlab en su versión 7.9 R2009b (software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio).

El Visual C++ se ha utilizado como entorno de programación para los códigos desarrollados en C++ ya que es el propuesto por los desarrolladores de Marilou. En este lenguaje se han programado las funciones que comandan el modelo de Marilou.

El programa Matlab se ha utilizado para el desarrollo del algoritmo antibalanceo que se aplica al modelo, así como para crear una interfaz de visualización de resultados y control de la simulación.

Para la consecución del objetivo principal de evaluar la idoneidad de la herramienta Marilou como herramienta de análisis de algoritmos antibalanceo en puentes grúa, deben cumplirse también los siguientes subobjetivos.

- Estudiar el entorno de modelado y simulación de Marilou y las posibilidades que ofrece.
- Desarrollar el modelo de una grúa puente con la herramienta Marilou.
- Definir e integrar en el modelo los sensores y actuadores que sean necesarios para implementar un control antibalanceo.
- Estudiar el algoritmo antibalanceo propuesto y desarrollar un caso sencillo para probar en el modelo.

1.3. Estructura del documento.

El presente Proyecto Fin de Carrera se divide en seis capítulos, bibliografía y anexos.

CAPÍTULO 1. INTRODUCCIÓN.

En este capítulo se hace una breve introducción al proyecto TechnoFusión, que ha motivado el desarrollo del proyecto del presente documento. Y a continuación se enumeran los objetivos del proyecto y se define la estructura que seguirá el mismo.

CAPÍTULO 2. MARILOU.

En este apartado se estudia, en primer lugar, las características del programa de modelado y simulación Marilou. A continuación, se desarrollan las funciones básicas disponibles en Marilou y se describe todo el proceso que se ha realizado para construir el modelo de la grúa puente y su posterior simulación.

CAPÍTULO 3. INPUT SHAPING.

En el Capítulo 3, en base a lo propuesto por Garrido y Abderahim del Robotics Labs de la Universidad Carlos III de Madrid, en su artículo *"Anti-swinging Input Shaping Control of an Automatic Construction Crane"* [2] y los estudios previos realizados por Singh y Singhose [4], se estudiará el algoritmo

de control *"Input Shaping"* y se buscará un ejemplo sencillo que aplicar sobre el modelo para comprobar su funcionamiento.

CAPÍTULO 4. INTERFAZ GRÁFICA DE CONTROL Y VISUALIZACIÓN. MATLAB.

En él se describe el entorno gráfico que se ha realizado en Matlab para interactuar con el modelo. Además, se explican las funciones creadas para aplicar el control antibalanceo.

CAPÍTULO 5. RESULTADOS.

En este capítulo se presentan los resultados obtenidos de oscilación en la carga para las diferentes simulaciones realizadas, aplicando y sin aplicar el algoritmo de control al modelo de la grúa.

CAPÍTULO 6. CONCLUSIONES.

Aquí se estudian los resultados obtenidos y se presentan las conclusiones a las que se ha llegado sobre el programa Marilou como herramienta para diseñar grúas de gran tonelaje con sistemas antibalanceo.

CAPÍTULO 2

MARILOU

2.1. El simulador

Marilou es un proyecto realizado por la empresa francesa Anikode, que nace para poder desarrollarlo. Este proyecto dispone de la ayuda europea al desarrollo de proyectos innovadores (SICE).

Marilou Robotics Studio es un entorno de modelado y simulación de robots, ya sean móviles, humanoides o brazos articulados, que puede operar en entornos que respeten las leyes de la física del mundo real. Es una avanzada herramienta con una técnica de diseño, estudio y control que servirá a la próxima generación de robots.

Éste posibilita construir los diseños más complejos desde cero y probarlos en un entorno gráfico realista, con un simulador en tiempo real, que permite experimentar el comportamiento de los algoritmos directamente. Además, en el entorno de simulación 3-D se puede ver cómo los sensores y actuadores reaccionan a las propiedades físicas de los objetos.

Los robots son máquinas costosas y complejas que, a veces, son muy difíciles de programar y, además, hay infinidad de situaciones que no se prevén durante la programación, sin contar con el riesgo de dañar un robot durante su fase de desarrollo. Por ello, a menudo es mejor usar entornos virtuales de simulación, donde los errores cometidos durante la fase de desarrollo no son muy graves. Los simuladores de robótica ofrecen, por lo tanto, ventajas significativas en un entorno controlado.

Marilou, al ser un simulador dirigido específicamente al diseño de robots, debe ser capaz de representar el mundo real y llevar a cabo simulaciones para estudiar el movimiento y las reacciones del robot en un entorno y un tiempo. En Marilou, el simulador permite a los desarrolladores crear tanto entornos como robots, lo que posibilita que se pueda probar diversas situaciones.

Por ejemplo, si queremos crear un robot encargado del montaje de palés para reducir los costes de operación, debemos calcular la velocidad óptima de cada tarea y encontrar una trayectoria que responda a las especificaciones, teniendo en cuenta que el robot está sujeto a unas restricciones mecánicas y estructurales, así como a factores del entorno, como pueden ser las colisiones.

Así, es mejor probar diferentes trayectorias en un simulador donde se pueda cometer errores sin dañar los equipos reales. Una vez conseguida la trayectoria óptima y depurados los posibles errores, se puede probar con menos riesgo en el equipo real.

Sin embargo, es importante recordar que, para que el simulador sea eficiente, el modelo que se use debe ser funcional y estructuralmente correcto, es decir, debe ser construido de manera que los resultados obtenidos y las predicciones del simulador sean relevantes para la tarea que se intenta llevar a cabo.

Por tanto, para el desarrollo de un proyecto en Marilou es importante poder simular el control antes de implementarlo en un entorno real, debido al tamaño de las estructuras y de la sensibilidad del trabajo a realizar.

Por otro lado, el editor de Marilou es completamente gráfico, lo que permite crear robots más complejos con relativa facilidad. Éstos se podrán ejecutar después en uno o varios mundos para probar los algoritmos que se hayan creado.

Con el fin de poder crear un entorno lo más real posible, Marilou dispone de una amplia gama de materiales predefinidos que se pueden representar en las superficies del robot o de cualquier otra entidad sujeta a las leyes de la física, como son: mesas, sillas, suelos…

Asimismo, el editor dispone, por un lado, de sensores y actuadores que se pueden asociar a los módulos del robot o sus articulaciones y, por otro, de formas geométricas predefinidas con las que se puede recrear cualquier modelo. Estas formas se ensamblan entre sí formando cuerpos rígidos o articulaciones.

Así desarrollados, los módulos de ejecución de Marilou se cargan en simuladores de tiempo real para poder evaluar el modelo en mundos virtuales con condiciones reales.

2.2. Funciones básicas de Marilou.

Marilou funciona mediante proyectos. El primer paso para crear la simulación es la elaboración de un proyecto (figura 2.1). Este proyecto contiene todas las entidades físicas que componen el modelo, así como la dependencia con respecto a recursos externos o entidades físicas. El proyecto también contiene las configuraciones de simulación.

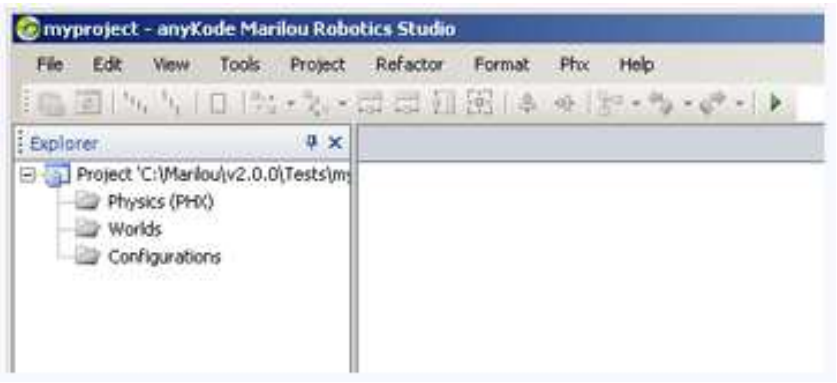


Figura 2.1: Estructura base de un proyecto de Marilou.

Para empezar a elaborar un modelo se debe crear el entorno, conocido como *"world"* (o mundo). Un mundo es la entidad de más alto nivel dentro de Marilou. Se puede tener varios mundos por cada proyecto, aunque solo se puede simular uno cada vez. Al crear un mundo, Marilou muestra cuatro vistas que se utilizarán para crear los modelos (figura 2.2).

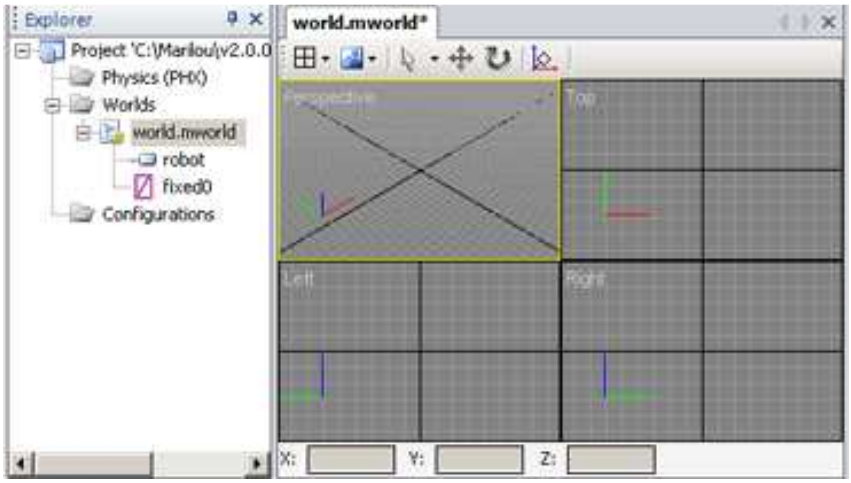


Figura 2.2: Cuatro vistas de un mundo para realizar el modelo.

Otro elemento imprescindible para la ejecución de la simulación es la configuración (o escenario). Una configuración es un formulario que contiene varios parámetros de simulación, incluyendo el *".mworld"* que va a ser utilizado o los programas para controlar los sensores y actuadores. Un proyecto puede tener varias configuraciones para simular diferentes situaciones o algoritmos para resolver algún problema específico (figura 2.3).

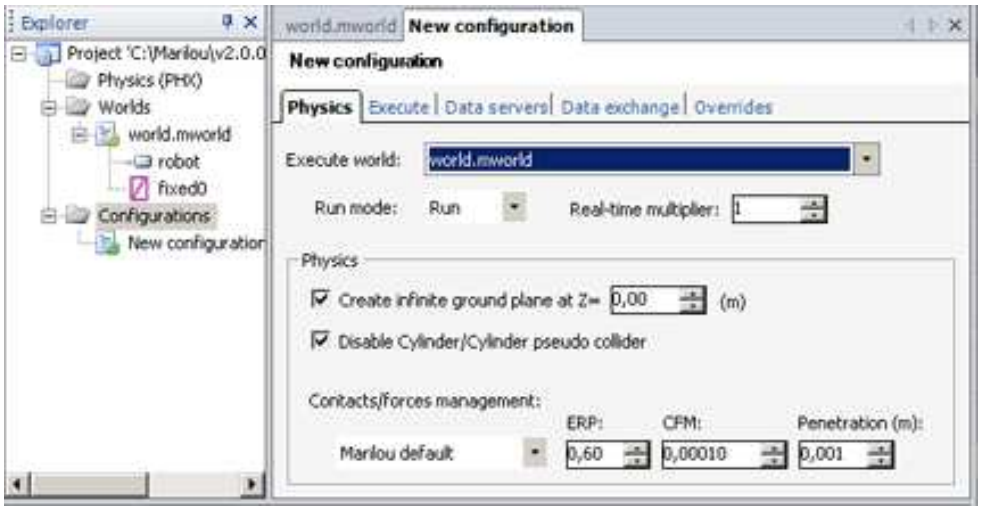


Figura 2.3: Formulario de configuración de la simulación.

El modelo físico se puede crear a partir de elementos predefinidos en Marilou o creando modelos previos de piezas más complicadas para después ensamblarlos. Estos modelos se llaman PHX. Para crearlos se dispone de los menús *"Add"*, *"Modify"* y *"Display"*.

En el panel *"Add"* (figura 2.4) se encuentran los objetos que se utilizan para formar el modelo físico. Hay tres opciones: *"Physics"*, *"3D"*, *"Link"*.

"Physics" (figura 2.4.a): Se utiliza para crear los modelos de colisión. Con estos modelos físicos se crea el elemento que se desea simular, ya sean paredes o partes de un robot. Pueden ser estáticos si van a actuar como un muro o un obstáculo, o dinámicos si se quiere que realicen algún tipo de movimiento.

Por un lado están los objetos nativos *("native Physics")* que son formas geométricas sencillas y predefinidas en Marilou para conseguir una mayor eficiencia en la simulación según el modelo de colisión de Marilou.

Y por otro, se tiene el grupo *"Triangle Meshes"*. Es un modelo de representación de superficies que se utiliza para poder representar, por medio de una malla de triángulos, una superficie compleja. Se recomienda usar formas nativas siempre que sea posible ya que, aunque se dispone de objetos optimizados, es más lento al simular que si solo se utilizan formas nativas, pues la posibilidad de una colisión se debe calcular con todos los triángulos que representan la superficie.

Además, desde este panel también se pueden agregar PHX ya creados y ensamblarlos o usarlos en el modelo. De esta forma, si hay un objeto que se va a repetir muchas veces en un proyecto, es conveniente crear un PHX de ese objeto para luego cargarlo en el proyecto cuantas veces se quiera.

"3D" (figura 2.4.b): Se usa para dar realismo visual a la simulación o presentar gráficos *("Render Geometries")* y para marcar las zonas de influencia de cámaras o sensores y establecer puntos de luz *("Render utilities")*, pero no afecta en sentido físico a la simulación.

"*Link*" (figura 2.4.c): Está dividido en dos partes, *"Rigid body"* y *"Joints"*.

"Rigid Body" o cuerpo rígido es un elemento esencial para componer el modelo. Establece una unión rígida y dinámica entre dos o más objetos físicos *("physics")* convirtiéndolos en dinámicos. Para ello Marilou calcula los esfuerzos que se presentan en las uniones y el centro de gravedad del conjunto. Así los cuerpos estáticos consumen menos CPU que los dinámicos y se recomienda que cuando sea posible, el cuerpo se defina como estático.

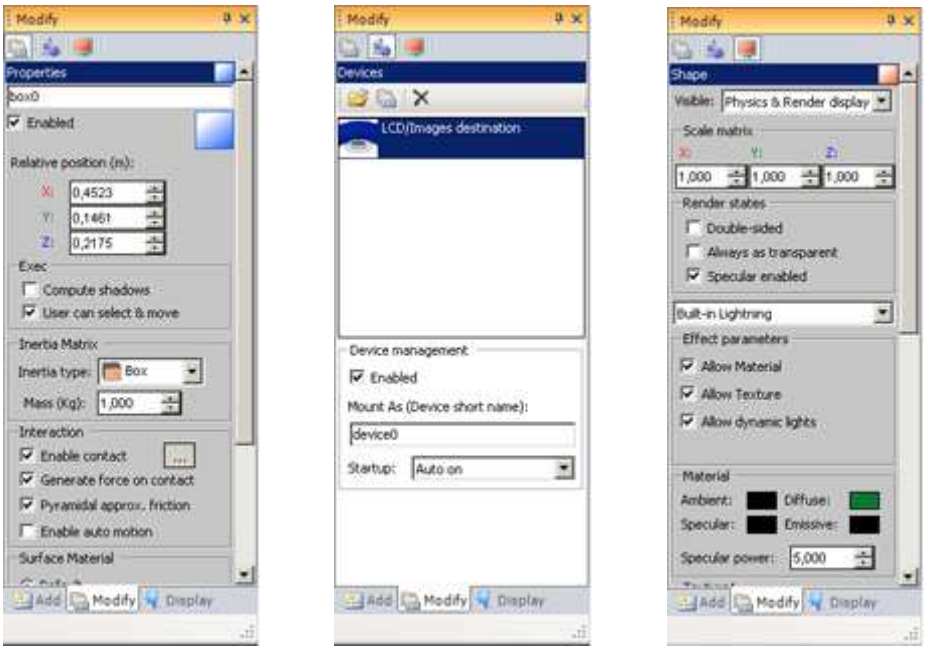
Los *"Joints"* o uniones determinan los grados de libertad que tiene un cuerpo rígido respecto a otro. En las propiedades de estos elementos se puede definir el sentido de los movimientos, así como establecer límites a los mismos. No tienen por que unir dos cuerpos rígidos, se puede establecer la unión de un cuerpo rígido con un entorno estático.

(a) (b) (c)

Figura 2.4: Panel *"Add"*: (a) *"Physics"*, (b) *"3D"*, (c) *"Link"*.

El panel *"Modify"* (figura 2.5) permite acceder a diferentes parámetros de los PHX (subpanel *"properties"*. Figura 2.5.a), así como asignar dispositivos (subpanel *"devices"*. Figura 2.5.b) como motores, sensores y/o cámaras, a las

geometrías que conforman el modelo y modificar las características visuales de los elementos *"3D"* (subpanel *"shape"*. Figura 2.5.c).



(a) (b) (c)

Figura 2.5: Panel *"Modify"*: (a) *"Properties"*, (b) *"Devices"*, (c) *"Shape"*.

El panel *"display"* (figura 2.6) permite seleccionar qué elementos, tanto de los *"worlds"* como de los PHXs, se deben visualizar. Se puede seleccionar ver los ejes de todos los elementos o solo el eje principal, y/o ver o no los cuerpos rígidos.

Figura 2.6: Panel *"Display"*.

Los *"devices"*, o dispositivos integrados, permiten al modelo percibir lo que hay a su alrededor e interactuar. Marilou dispone de elementos predefinidos como son: motores, servos, sensores de distancia, giroscopios, cámaras…

Estos dispositivos deben ir siempre asociados a algún elemento del modelo. Así, un sensor de distancia debe ir asociado a una zona y un motor al eje de una articulación o unión *("joint axis")*. Cada uno tiene su propio panel de propiedades donde se podrá configurar en función del uso que se le vaya a dar. Además, disponen de funciones predefinidas en MODACPP que se utilizarán para definir su comportamiento.

Con estos menús se termina de conformar el modelo físico. Si se quiere probar un control o programar el funcionamiento de los sensores actuadores se debe recurrir a los *"wizards"* o asistentes.

Se dispone de tres asistentes diferentes: MATLAB, MODA (hace referencia a paquetes de programación en C++ y C#) y el RT-Maps.

Estos asistentes sirven tanto para definir nuevos dispositivos como para diseñar los controles o programas que se desean probar sobre el sistema.

Los programas o controles realizados se deben incluir en el panel *"configurations"* donde se definen los diferentes parámetros de ejecución: gravedad, tiempo de ejecución, algoritmo a ejecutar, etc.

2.3. Modelado de la grúa.

Para poder probar algoritmos de control se va a crear en el simulador MARILOU un modelo de una grúa puente.

La grúa se ha modelado usando formas sencillas (cubos, paralelepípedos, esferas, cilindros) para conseguir un modelo aproximado. Se ha diseñado a partir de varios módulos: la base, el carro longitudinal, el carro transversal, la viga y la carga (figura 2.7 y 2.8).

Figura 2.7: Grúa real referencia para el modelo.

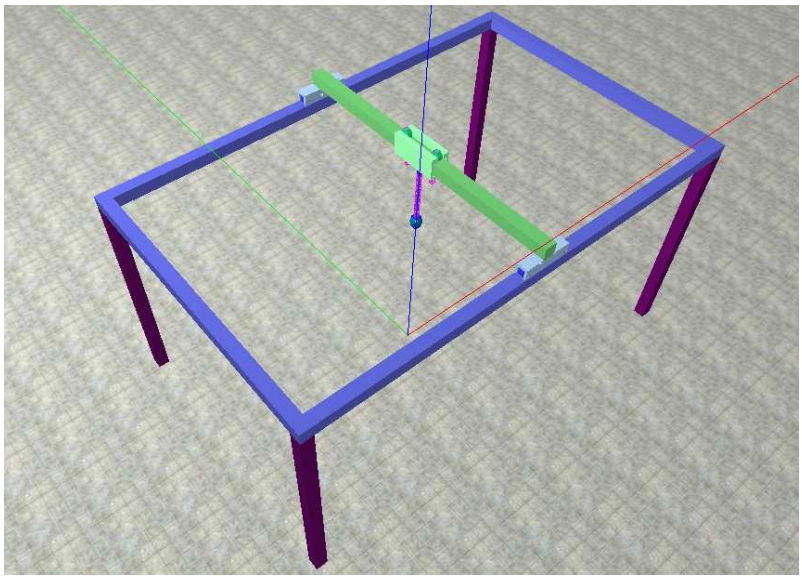


Figura 2.8: Modelo de la grúa

La base (figura 2.9): Está compuesta por cuatro paralelepípedos de dimensiones 0.2x0.2x3.8 que actúan como patas, dos paralelepípedos de 8x0.2x0.2 para el largo de la estructura y dos de 0.2x4.6x0.2 para el ancho.

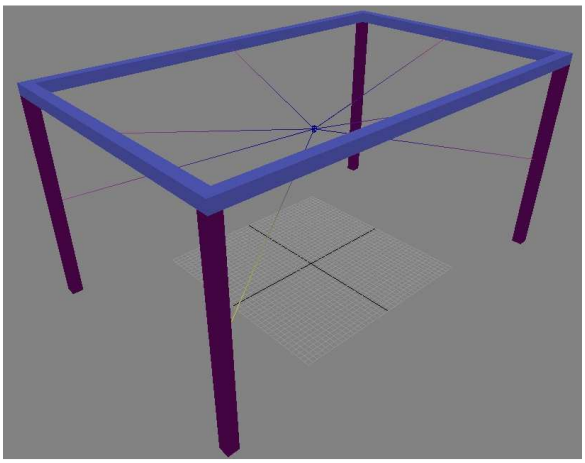


Figura 2.9: Base.

Para crear estos paralelepípedos se ha usado el elemento *"Box"* del subpanel *"Physics"*, panel *"Add"* (Panel detallado en el apartado 2.2)

Todos estos paralelepípedos se han asociado a un mismo cuerpo rígido (subpanel *"Link"* del panel *"Add"* detallado en el aparatado 2.2). De esta manera todos los elementos actúan como uno solo, formando una única estructura.

Carro longitudinal (figura 2.10): En primer lugar se ha creado una estructura en forma de "U" invertida formada por tres paralelepípedos, dos de dimensiones 0.04x0.8x0.18 y uno de 0.12x0.8x0.04 metros, que actúan como cuerpo del carro longitudinal. Estos tres elementos están asociados a un mismo cuerpo rígido para que actúen como uno solo.

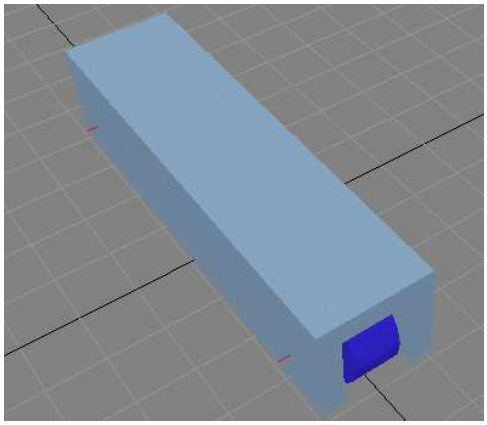


Figura 2.10: Carro longitudinal

Para modelar las ruedas se han creado dos cilindros de radio 0.08 m. y largo 0.08 m. (elemento *"Cylinder"*, subpanel *"Physicis"*, panel *"Add"*) cada uno de ellos asociado a un cuerpo rígido.

Para unir cada rueda con el cuerpo del carro se ha utilizado la unión *"Hinge"* (subpanel *"Link"*, panel *"Add"*). Este tipo de unión permite un movimiento de rotación de un cuerpo rígido respecto a otro. En este caso, se ha configurado la unión de tal manera que se permita la rotación del carro (/body0 en la figura 2.11) respecto del eje de la rueda (/body2 en la figura 2.11).

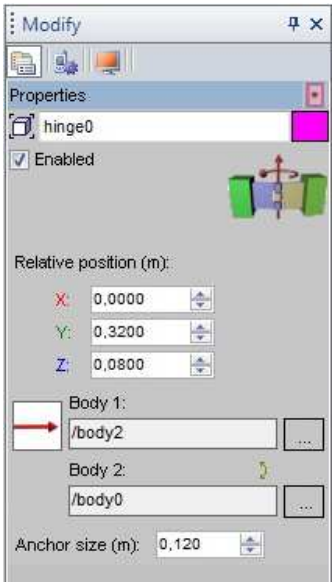


Figura 2.11: Panel de definición de la articulación *"Hinge"*.

El elemento *"Hinge"* establece una unión que permite un grado de libertad pero, para que se produzca el movimiento hay que usar un actuador.

En este caso se ha seleccionado un motor que actúa sobre el eje de la rueda (eje de Hinge2 en la figura). Este elemento se añade pulsando en el botón "…" que aparece en el panel de definición de la unión *"Hinge"* que se puede ver en la figura 2.11.

Todo este conjunto se ha guardado como un PHX para agregarlo al modelo final como una única pieza.

Carro transversal (figura 2.12): Este carro tiene una estructura diferente al anterior, ya que de él va suspendida la carga. Consiste en una U de mayor tamaño que la del carro longitudinal, que se coloca en torno a la viga dejando que las

ruedas se apoyen en la parte superior de ésta y la parte plana del carro quede suspendida por debajo. De esta última parte es de donde se sujeta la carga.

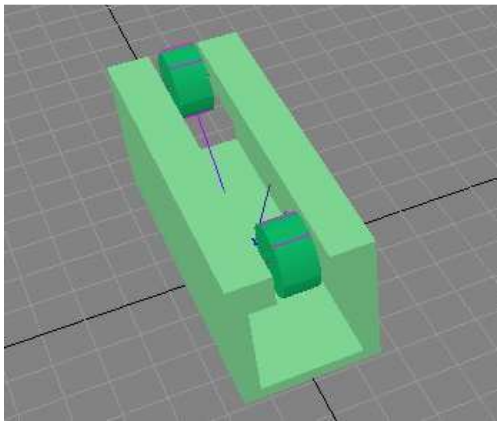


Figura 2.12: Carro transversal.

Esta estructura está formada por un paralelepípedo de dimensiones 0.32x0.8x0.03 m., que es la superficie plana de la que se suspende la carga, dos paralelepípedos de 0.04x0.8x0.24 m. para los laterales del carro y dos de 0.1x0.8x0.08 m. que quedan por encima de la viga sujetando las ruedas. Estos cinco elementos están asociados a un mismo cuerpo rígido y forman el cuerpo del carro transversal.

Para modelar las ruedas también se han creado dos cilindros de radio 0.08 m. y largo 0.08 m., cada uno de ellos asociado a un cuerpo rígido. Igual que se hizo en el carro longitudinal, estos cilindros se unen al cuerpo del carro usando el elemento *"Hinge"*.

En la articulación *"Hinge"* de una de las ruedas se ha añadido un motor, igual que se hizo para el carro anterior.

Todo este conjunto se ha guardado como un PHX para agregarlo al modelo final como una única pieza.

La viga (figura 2.13): El módulo de la viga está compuesto por dos carros longitudinales y un paralelepípedo alargado que actúa como viga. El paralelepípedo alargado, de la misma longitud que el ancho de la estructura base, se apoya sobre los dos carros longitudinales.

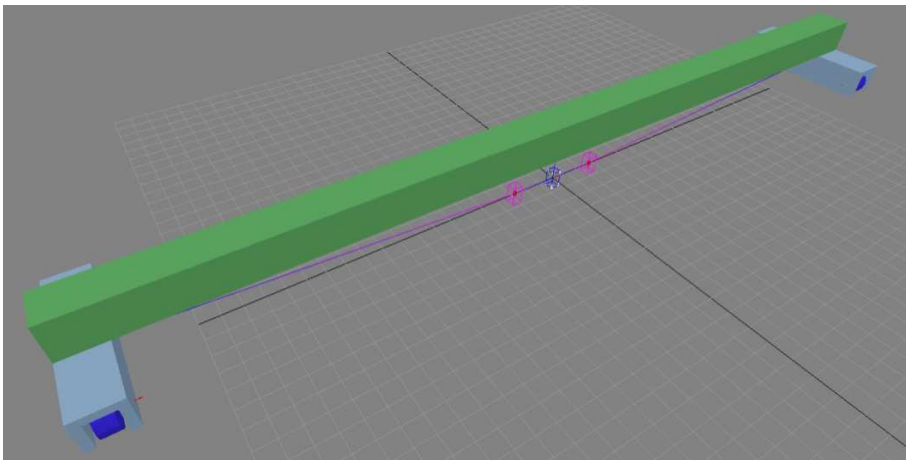


Figura 2.13: Viga.

En este caso se ha creado un paralelepípedo para la viga asociado a un cuerpo rígido.

Para agregar los carros longitudinales se debe seleccionar *"Physics File"* en el panel *"Add"*, subpanel *"Physics"*, sección *"Physics Assembles* (PHX)". Esto permite seleccionar elementos "PHX" ya creados.

Se seleccionan dos carros longitudinales que se colocan a cada extremo de la viga. Para unir cada carro longitudinal se ha usado una unión *"fixed"* (panel *"Add"*, subpanel *"Link"*).

Este conjunto se guarda como un PHX para poderlo agregar en conjunto al modelo final.

La carga: La carga se modela como una pequeña esfera (elemento *"Sphere"*, subpanel *"Physics"*, panel *"Add"*) con la masa correspondiente con la que se quiera simular el algoritmo. Esta esfera representa cualquier carga con una forma más compleja y la misma masa que tenga su centro de gravedad donde se ubica la esfera.

Esta esfera se ha definido como un cuerpo rígido al que se le ha asociado un giroscopio y un sensor de distancia.

El giroscopio se agrega a la esfera en el panel *"Modify"*, subpanel *"Devices"*. Es uno de los dispositivos predefinidos en el programa con el nombre: *"Gyrometer/Accelerometer/Gyroscope"* (figura 2.14). En el panel de configuración del dispositivo se activa, para todos los ejes (x, y, z), la función de giroscopio *("gyroscope")*, mientras que se dejan inactivas las funciones de acelerómetro y girómetro (figura 2.15).

Figura 2.14: Panel *"Modify"*, subpanel *"Devices"*. Se agrega el giroscopio.

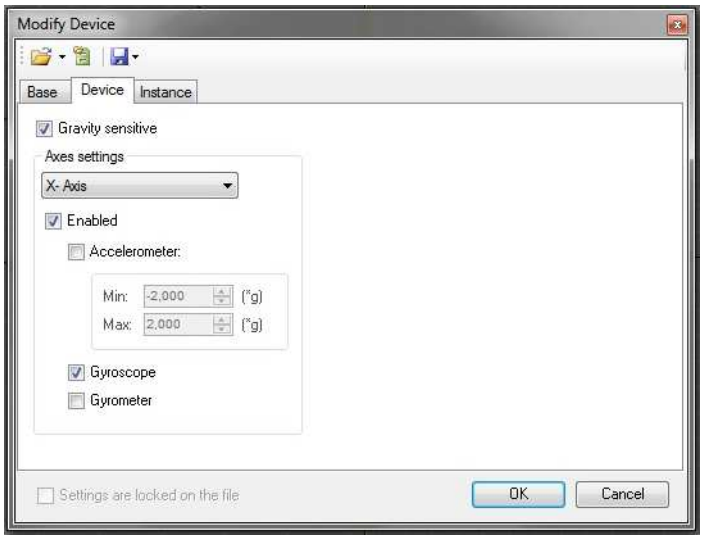


Figura 2.15: Panel de configuración del dispositivo *"Gyrometer/Accelerometer/Gyroscope"*

Para definir un sensor de distancia es necesario haber creado previamente una zona de influencia. En este caso se ha asociado a la esfera una zona de un metro de longitud con una abertura de 20º. Estos parámetros se han elegido de forma arbitraria aunque teniendo en cuenta que la apertura no debe ser muy

grande para que, en los momentos en que la carga esté cerca de alguna de las cuatro patas, éstas no causen interferencia en la medida. En la figura 2.16 se puede ver la zona representada por una sombra verde.

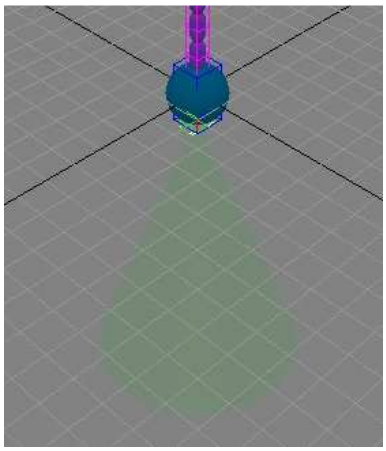


Figura 2.16: Carga-esfera y zona asociada a ella.

A esta zona se le asigna el sensor de distancia que se encuentra entre los dispositivos predefinidos en Marilou (Panel *"Modify"*, subpanel *"Devices"*, dispositivo *"Distance"*) y se configura para que sea un sensor de distancia mediante ultrasonidos (figura 2.17).

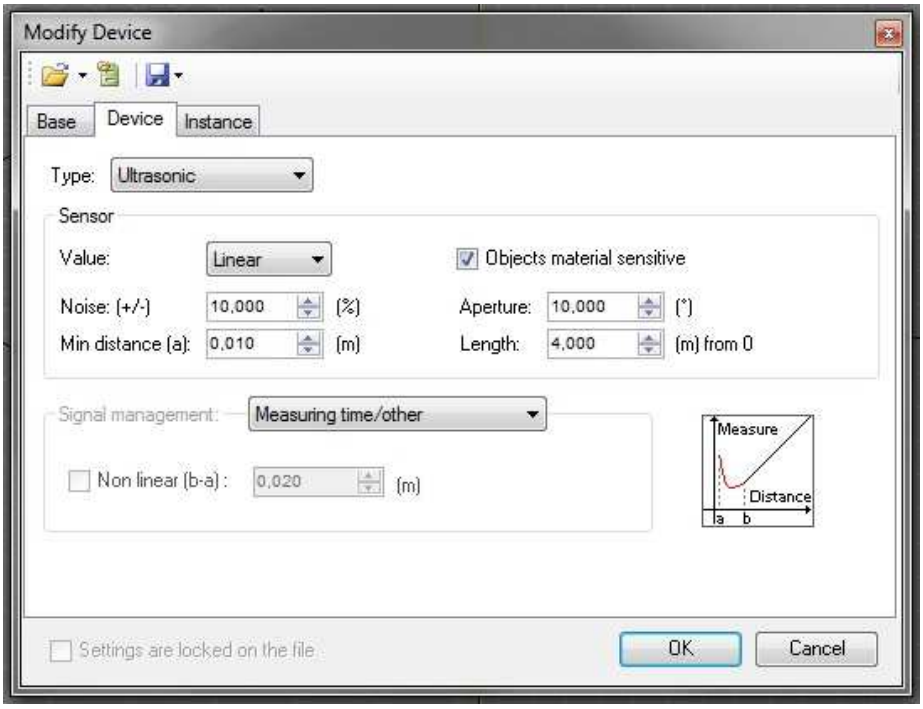


Figura 2.17: Panel de configuración del dispositivo *"Distance"*.

Dado que uno de los objetivos del proyecto es simular el comportamiento de una carga que esté suspendida de la grúa mediante un cable o cuerda, es necesario que el modelo represente las características dinámicas de estos elementos. Cuanto mejor esté modelado más fiable será el resultado de la simulación, ya que influirá directamente en la oscilación de la carga durante el movimiento.

No hay ningún elemento que simule estas características por sí solo, así que se ha modelado la cuerda de dos formas diferentes para poder comparar.

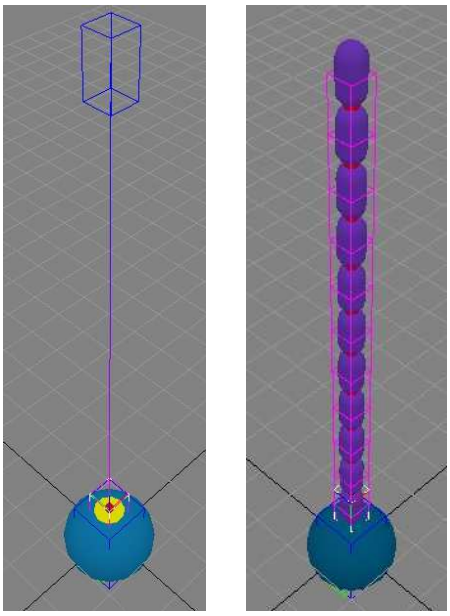
Cuerda 1 (Figura 2.18.a): Consiste en un elemento rígido, sin masa, (elemento *"Ray"*, subpanel *"Physics"*, panel *"Add"*) que une el carro y la carga con rótulas en los extremos. Las rótulas (elemento *"Ball"*, subpanel *"Joints"*, panel *"Add"*), como nexos de unión, permiten que la dinámica se parezca a la de una cuerda, aunque al tratarse de un elemento rígido, sin flexión, no puede simular de forma exacta esa cuerda.

Cuerda 2 (Figura 2.18.b): Para conseguir mayor flexibilidad en la cuerda se ha creado un segundo modelo, en el que ésta se forma uniendo una especie de eslabones de cadena. Cada eslabón está compuesto por una cápsula (elemento *"Capsule"*, subpanel *"Physics"*, panel *"Add"*) de pequeña longitud y una rótula que permite su unión con el siguiente elemento.

Uniendo a través de rótulas varios eslabones entre sí y, a su vez, los extremos con la carga y el carro transversal, se consigue un modelo de cuerda que simula mejor la dinámica de la misma.

Tras varias pruebas se ha comprobado que no se puede usar cápsulas muy pequeñas porque al simular se convierte en un modelo inestable y rompe. Esto también ocurre si se aumenta mucho la masa de la carga (una carga superior a 100 kg).

Se ha conseguido usar este tipo de cuerda limitando el número máximo de cápsulas a 11 de 0.1 m de longitud cada una y la masa máxima de la carga a 100 kg.



(a) (b)

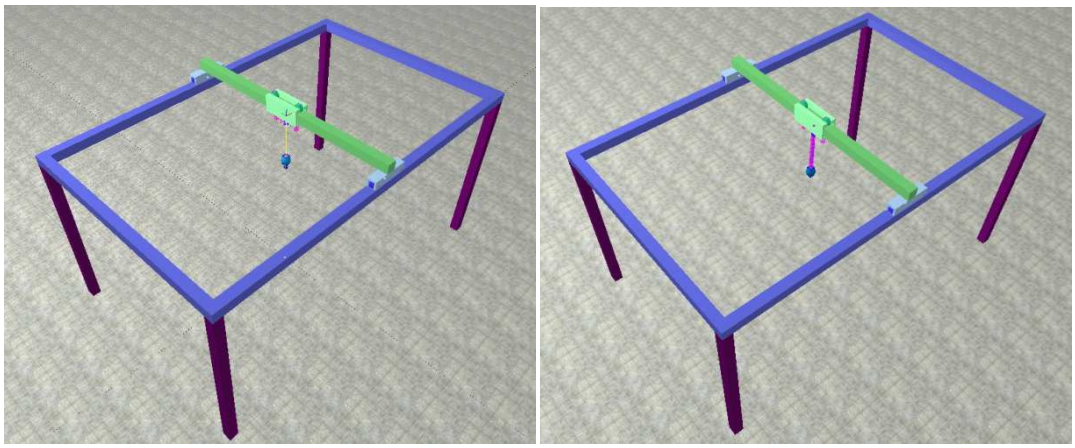
Figura 2.18: (a) Carga 1: La esfera cuelga de un elemento rígido, sin masa.

(b) Carga 2: La esfera cuelga de una cadena formada por once cápsulas unidas por rótulas.

Una vez que se han modelado todos los elementos, deben unirse para formar el modelo completo de la grúa puente. Cada elemento de la grúa se ha guardado como un PHX, lo que permite que se pueda usar tantas veces como se quiera.

Se coloca la viga de tal manera que los carros longitudinales puedan deslizarse sobre el carril de la base.

El carro transversal se coloca dejando la viga en el hueco formado por la U y las ruedas, de tal forma que éstas actúen sobre la parte superior para mover el carro. De la parte inferior de este carro cuelga la carga (figura 2.19).



(a) (b)

Figura 2.19: (a) Modelo de grúa con Carga 1. (b) Modelo de grúa con Carga 2.

2.4. Sensores y actuadores.

Después de modelada la estructura de la grúa, es necesario modelar los sensores y actuadores que permiten un funcionamiento similar al real.

Se utilizan tres motores con encoder, uno en cada carro, para mover la carga. Para saber la longitud de la cuerda en cada momento, se ha definido un sensor de distancia que mide el espacio que hay de la carga al suelo. Además, se ha colocado un giroscopio en la carga que indica la oscilación de la misma para poder comprobar que el algoritmo funciona correctamente.

Los motores están ubicados en una de las ruedas de cada uno de los carros y se controlan asignando consignas de velocidad. Si se desea un movimiento lateral de la carga se asignará una velocidad al motor del carro transversal y si lo que se quiere es un movimiento longitudinal se le asignará a los motores de los dos carros longitudinales. El tipo de señal recibida por los motores viene definido por el algoritmo de control, y será el que defina la oscilación final de la carga.

Se puede definir los parámetros principales del motor en un panel de propiedades como el de la figura 2.20.

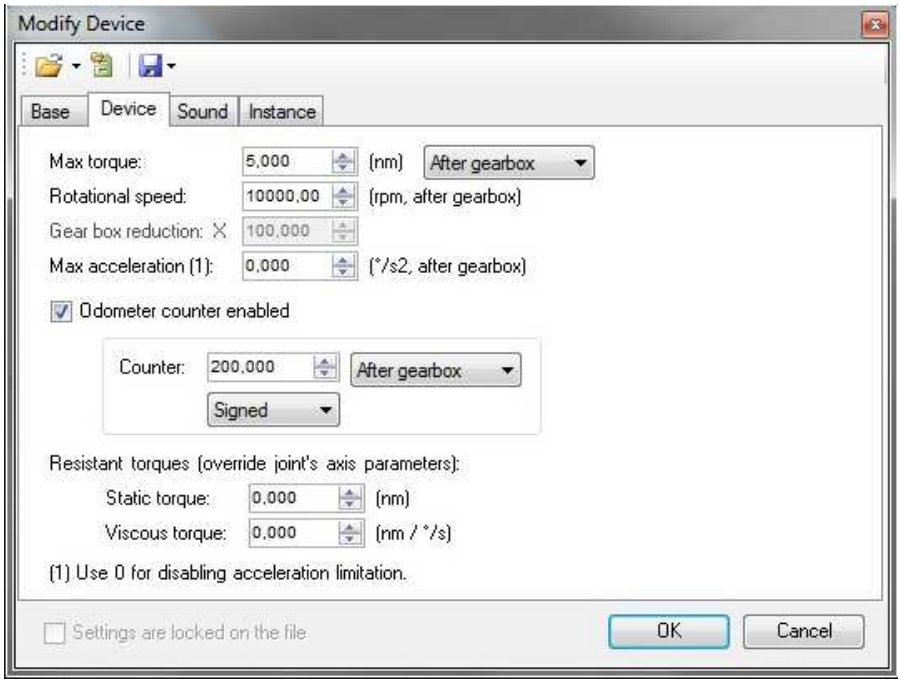


Figura 2.20: Panel de configuración del motor.

En este cuadro se puede definir la máxima velocidad y aceleración, el valor de la reductora o si está activo o no el encoder. En este modelo todos los motores disponen de encoder para poder conocer en cada momento la posición de la carga. Los tres motores se han definido con las mismas características.

Además, en la esfera que representa la carga de la grúa se ha dispuesto un sensor de distancia mediante ultrasonido y un giroscopio.

El sensor de distancia por ultrasonido se usa para conocer la longitud de la cuerda que sujeta la carga. Para definirlo ha sido necesario crear una zona de influencia, que determina el alcance del sensor (sombreado en verde en la figura 2.21).

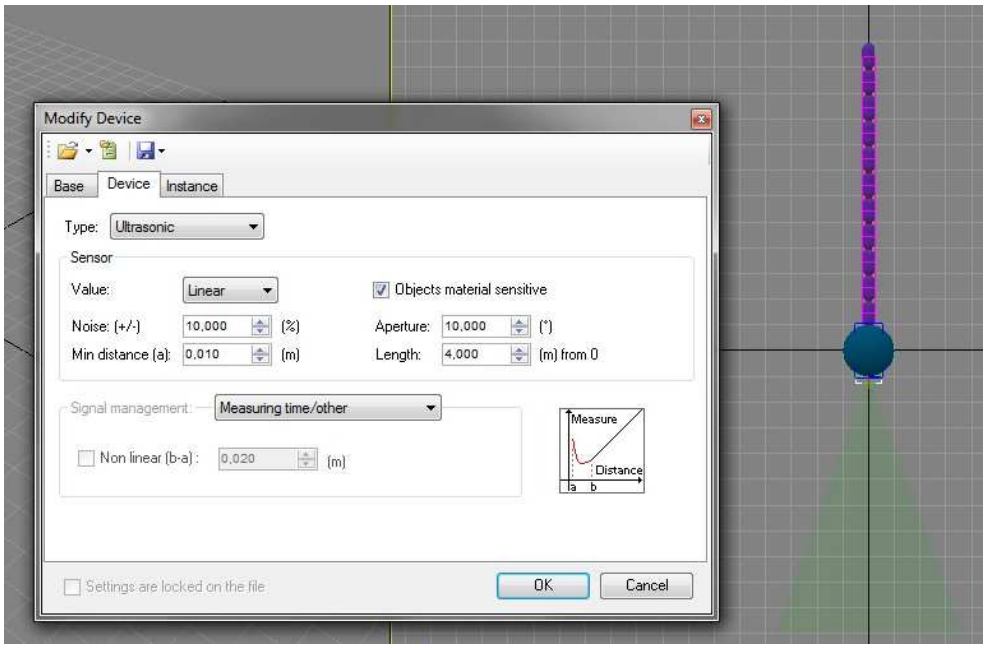


Figura 2.21: Panel de configuración del ultrasonido, zona y carga.

En el panel de propiedades de este dispositivo se define el alcance y el ángulo de apertura del sensor.

Esto se dispuso así porque la idea inicial era poder variar la longitud de la cuerda durante la simulación y así poder explorar más aspectos del algoritmo. Marilou no permite esta variación, por lo que la longitud de la cuerda se mantiene constante. Aun así, se ha mantenido el elemento medidor por si en un proyecto futuro se pudiera implementar esta función.

En MARILOU existe un dispositivo que integra tres funciones: girómetro, acelerómetro y giroscopio, que se pueden habilitar o deshabilitar para cada eje de forma independiente. En este caso sólo es necesaria la función como giroscopio, que nos permite medir los ángulos para comprobar si funcionan los algoritmos que se probarán posteriormente. Se ha definido así en el panel de propiedades para cada eje (Figura 2.22).

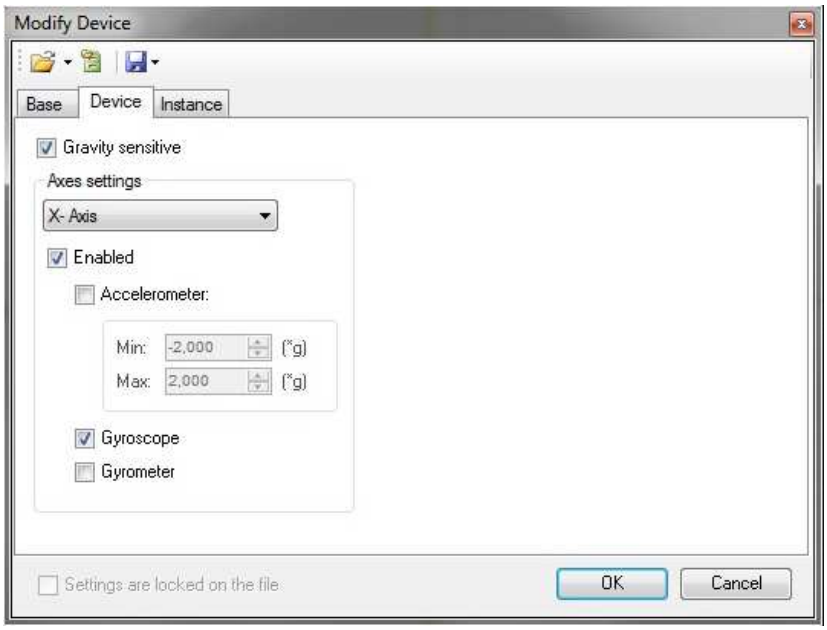


Figura 2.22: Panel de configuración del giroscopio.

El giroscopio está situado en la esfera y devuelve los ángulos de inclinación de la misma respecto a los ejes fijos de la base. Representando estos ángulos para diferentes configuraciones de velocidad se puede evaluar la efectividad del algoritmo.

2.5. Programación.

Una vez realizado el modelo, se debe definir el modo de funcionamiento que hay detrás de cada sensor y actuador para que se pueda ejecutar un algoritmo de control sobre la oscilación de la carga. En este caso se ha optado por un programa combinado que usa tanto las librerías MODA (paquetes de programación en C++ de Marilou) como librerías para la comunicación con Matlab (*"mex.h")*.

2.5.1. MODA. C++

Marilou dispone de una librería de C++: Modacpp.h, donde hay clases predefinidas para el funcionamiento de los distintos sensores y actuadores. También te permite añadir o crear elementos diferentes en caso de que fuera necesario.

En este caso se ha optado por utilizar las funciones predefinidas, ya que se trata de una primera aproximación para la comprobación del algoritmo.

Lo primero que se debe hacer es crear el modelo en el programa. Para eso se debe definir las clases predefinidas por Modacpp.h que configuran los diferentes elementos del modelo *("private")* y las que se han creado para la manipulación del mismo *("public").*

```
class MyRobot
        {
        private:
                RobotPHX *_pPhx;

                DeviceMotor *_pMotor0;
                DeviceMotor *_pMotor1;
                DeviceMotor *_pMotor2;
                DeviceAccelGyro *_pDevice0;
                DeviceDistance * _pDeviceDistance0;

        public:
                MyRobot::MyRobot   ( RobotPHX *pPhx);
                MyRobot::~MyRobot ();

                bool velocidad1(double vel);
                bool velocidad2(double vel);
                void MyRobot::datos (Moda::Commons::AXESXYZValues &a);
                void MyRobot::enconder (struct encodata &lectencoder);
                float distancia (void);
};
```

Las clases *"RobotPHX"*, *"DeviceMotor"*, *"DeviceAccelGyro"* y *"DeviceDistance"* vienen definidas en la librería Modacpp.h y corresponden al modelo principal, los motores, el giroscopio y el ultrasonido respectivamente.

A continuación se detallan las funciones definidas en *"public"* que han sido creadas para poder manipular el modelo.

*MyRobot::MyRobot(RobotPHX*pPhx)* inicializa el modelo, es decir, asigna a las diferentes clases que configuran el modelo la referencia de cada elemento.

```
MyRobot::MyRobot(RobotPHX *pPhx)
        {
        _pPhx=pPhx;

        _pMotor0=pPhx->QueryDeviceMotor("phx0/phx1/phx0/hinge0/a1/mototo");
        _pMotor1=pPhx->QueryDeviceMotor("phx0/phx1/phx1/hinge0/a1/mototo");
        _pMotor2=pPhx->QueryDeviceMotor("phx0/phx2/hinge0/a1/motor1");
        _pDevice0=pPhx->QueryDeviceAccelGyro("phx1/sphere0/device0");
        _pDeviceDistance0= pPhx->QueryDeviceDistance("phx1/zone0/distance");

        }
```

Se define, en primer lugar, el modelo "pPhx" y a continuación los sensores y actuadores que forman parte del mismo: el giroscopio *_pDevice0*, el sensor de ultrasonido *_pDeviceDistance0*, los motores de los carros longitudinales *_pMotor0* y *_pMotor1* y, para el del carro transversal, *_pMotor2*.

MyRobot::~MyRobot () elimina las conexiones del programa al modelo cuando se ha terminado de simular.

```
MyRobot::~MyRobot()
        {
        if(_pMotor0!=NULL)                  delete _pMotor0;
        if(_pMotor1!=NULL)                  delete _pMotor1;
        if(_pMotor2!=NULL)                  delete _pMotor2;
        if(_pDevice0!=NULL)                 delete _pDevice0;
        if(_pDeviceDistance0!=NULL)         delete _pDeviceDistance0;

        if(_pPhx) delete _pPhx;
        }
```

La clase que controla los motores está definida en las librerías de Marilou como *"DeviceMotor"*. Esta clase contiene diversos métodos que permiten obtener y fijar los parámetros que definen el funcionamiento del motor, como puede ser el valor de la reductora, la máxima velocidad y aceleración, o el valor del encoder.

Para definir la velocidad de los motores se ha creado las funciones *bool velocidad1(double vel)* y *bool velocidad2(double vel)*.

En este caso se utiliza el método *"SetVelocityDPS"*, para definir la velocidad de rotación del motor en cada momento. Este método fija al motor la velocidad que se le indica en grados por segundo, intentado aplicar el mínimo torque posible. La velocidad se define con la variable tipo *double "vel"*. Es un método predefinido por la librería Modacpp.h para la clase *"DeviceMotor"*.

```
bool MyRobot::velocidad2(double vel){
        bool bRet=false;
        if(_pMotor2!=NULL)
                {

                _pMotor2->SetVelocityDPS((float)vel);

                }
        return(bRet);
}

bool MyRobot::velocidad1(double vel){
        bool bRet=false;
        if((_pMotor0!=NULL)&&(_pMotor1!=NULL))
                {

                _pMotor0->SetVelocityDPS((float)vel);
                _pMotor1->SetVelocityDPS((float)vel);

                }
        return(bRet);
}
```

Para obtener el valor del encoder de los motores se ha definido la función *void MyRobot::enconder ("struct encodata &lectencoder").*

La clase "*GetOdometerCounter"* permite obtener el valor del encoder del motor. Con ese valor, el radio de la rueda y conociendo el punto de partida, se puede calcular la posición aproximada de la carga en cada momento.

Para almacenar el valor de los tres encoder se ha creado la estructura "*encodata"*, que guarda en una variable tipo double el valor de cada encoder.

```
void MyRobot::enconder (struct encodata &lectencoder){


        if((_pMotor0!=NULL)&&(_pMotor1!=NULL)&&(_pMotor2!=NULL))
        {

                lectencoder.b2=_pMotor2->GetOdometerCounter();
                lectencoder.b0=_pMotor0->GetOdometerCounter();
                lectencoder.b1=_pMotor1->GetOdometerCounter();

        }
        else
        {
                #ifdef DEBUGPRINT
                mexPrintf("odometer not found\r\n");
                #endif
        }

}

struct encodata{
        double b0;
        double b1;
        double b2;
};
```

void MyRobot::datos (Moda::Commons::AXESXYZValues &a) es la función que recoge las mediciones realizadas por el giroscopio. La clase que corresponde con el uso de este dispositivo es *"DeviceAcCelGyro"*. Los métodos definidos para esta clase permiten conocer las aceleraciones, inclinaciones y velocidad angular del dispositivo respecto a los ejes de base. Se puede ejecutar el método para cada eje por separado u obtener el valor de todos los ejes a la vez ("*GetXYZInstantValues"*). Estas mediciones siguen la estructura de la variable predefinida en Moda AXESXYZValues. En este caso, como sólo está activado el modo giroscopio, sólo habrá valores para las variables "*time"* y "*angles"*.

```
void MyRobot::datos (Moda::Commons::AXESXYZValues &a){


    if(_pDevice0!=NULL)
    {

        a=_pDevice0->GetXYZInstantValues();

        if(_pDevice0->GetLastSystemError()==MODA_EOK){   }
```

```
            else
            {
                #ifdef DEBUGPRINT
                mexPrintf("Error %d while reading values\r\n",_pDevice0->GetLastSystemError());
               #endif
            }

    }
    else
    {
            #ifdef DEBUGPRINT
            mexPrintf("accelerometer not found\r\n");
            #endif
    }

}

struct AXESXYZValues {

             float time;
             float LinearAccelerations[3];
             float AngularVelocities[3];
             float Angles[3];

};
```

La última clase creada *"float distancia (void)"* es para obtener las mediciones del sensor de distancia mediante ultrasonido. Éste está definido por la clase *"DeviceDistance"*. Esta clase tiene predefinidos dos métodos, los dos para obtener la distancia en metros al obstáculo más cercano; uno toma medidas de manera síncrona y el otro únicamente cuando se solicita. Se ha decidido usar *"GetMeasure"* que sólo toma la medida cuando se llama a la función. La distancia se almacena en la variable double *"dist"*.

```
float MyRobot::distancia (void){
   float dist;
   if( (_pDeviceDistance0!=NULL)  )
  {

      if(_pDeviceDistance0->GetLastSystemError()==MODA_EOK)
      {
        dist=_pDeviceDistance0->GetMeasure();
      }

      else
      {
        dist=0;
        #ifdef DEBUGPRINT
        mexPrintf("Error %d while reading values\r\n",_pDeviceDistance0->GetLastSystemError());
        #endif
      }
  }
  return dist;

}
```

Todas estas clases se encuentran en el archivo *"Robot.cpp"* adjunto en el anexo 3.

Una vez definidas todas las clases necesarias para controlar los sensores y actuadores del modelo, se crea un programa que las manipula. En este caso, para desarrollar el algoritmo de control que se quiere probar en el modelo, se ha creado un programa conjunto entre programación C++ y Matlab.

En C++ se ha desarrollado el archivo "cmex.cpp" (anexo 2) que incluye las conversiones necesarias de tipos de variable entre Matlab y C++ y llamadas a las clases que controlan los sensores y actuadores. Así mismo está preparado para responder a las acciones solicitadas desde Matlab con un código de banderas (tabla 1).

Bandera	Acción
'c'	Conectar con el modelo de Marilou
'd'	Desconecta el modelo de Marilou
'v'	Asigna velocidad a carros longitudinales
'V'	Asigna velocidad a carros transversales
's'	Recoge datos de oscilación del giroscopio
'e'	Hace una lectura de los encoders de los tres motores
'h'	Mide la distancia de la carga al suelo con el ultrasonido

Tabla 1: Código de banderas para comunicación de Matlab con el modelo.

Estos archivos se compilan creando el archivo *"Robot.dll"* al que accederá Matlab cuando quiera interactuar con el modelo desarrollado en Marilou. El desarrollo realizado en Matlab se verá en el Capítulo 4.

2.6. Simulación del modelo.

Una vez definido el modelo y todos sus elementos, solo queda simularlo. La simulación se configura con los paneles *"Configurations"*. Aquí se define el mundo y el algoritmo que se desea ejecutar, así como los parámetros de ejecución. Además, puede haber definidas configuraciones diferentes para un mismo modelo y seleccionar en cada momento cuál se quiere ejecutar.

Una vez configurada la simulación, se ejecuta pulsando el botón "*play*" que está en la parte superior de la ventana del programa. Al lado de este botón hay una ventana desplegable donde se puede seleccionar que "*Configurations*" ejecutar de entre los que se hayan preparado (Figura 2.23).

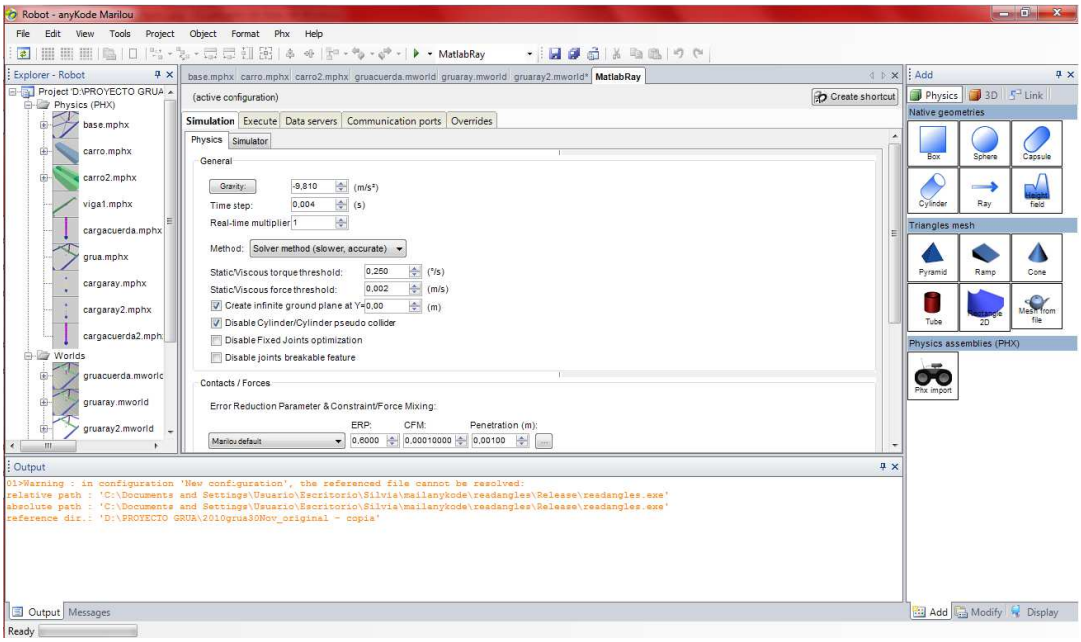


Figura 2.23: Ventana "*Configurations*" para la simulación del modelo.

Al pulsar el botón "*play*" se despliega una ventana donde se puede ver una perspectiva del modelo creado (Figura 2.24). En todos los casos se ha configurado la ejecución para que comience en pausa. Para activar definitivamente la ejecución del modelo se debe pulsar el botón "*play*" que hay, también, en la nueva ventana. Al hacerlo, se puede ver como el temporizador empieza a correr.

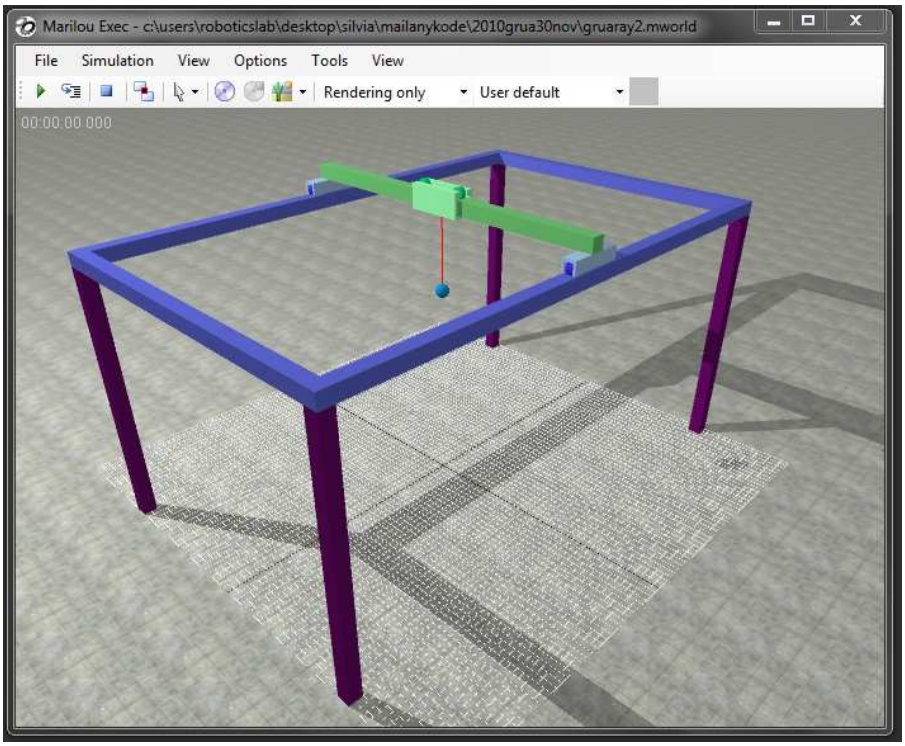


Figura 2.24: Ventana de simulación del modelo en Marilou.

Una vez está la ejecución en curso, se puede interactuar con ella desde el programa creado en Matlab (Capítulo 4).

CAPÍTULO 3

INPUT SHAPING

El objetivo principal de este proyecto es disponer de un entorno de simulación que permita comprobar la eficacia de diferentes algoritmos de control aplicables a puentes grúa de grandes dimensiones en tareas de movimiento de cargas pesadas. Como se explicó en el capítulo de introducción, este tipo de sistemas puede encontrarse en las instalaciones como las previstas en el proyecto TechnoFusión.

Para comprobar el modelo desarrollado se ha implementado el control de la grúa modelada por medio del algoritmo "*Input Shaping*"[2]. En este caso, lo que se pretende es controlar la oscilación de la carga de la grúa para que sea nula.

3.1. Introducción al método "*Input Shaping*"

Cuando la grúa se pone en movimiento, la carga actúa como un péndulo y empieza a oscilar (figuras 3.1 y 3.2). Lo mismo ocurre al detenerse. Esta oscilación en la carga viene determinada por la longitud del cable del que cuelga.

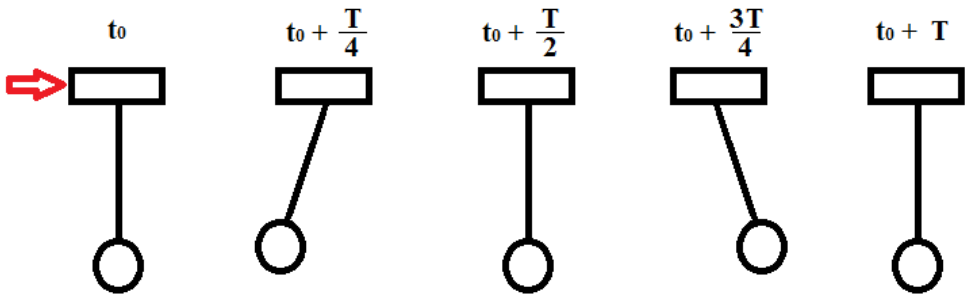


Figura 3.1: Oscilación de la carga ante un cambio de estado.

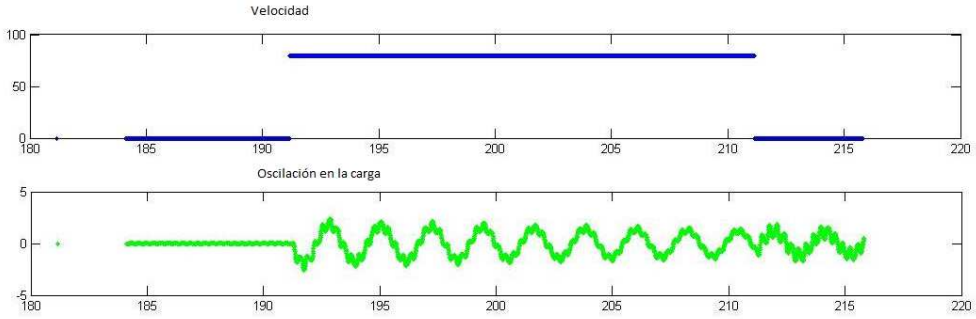


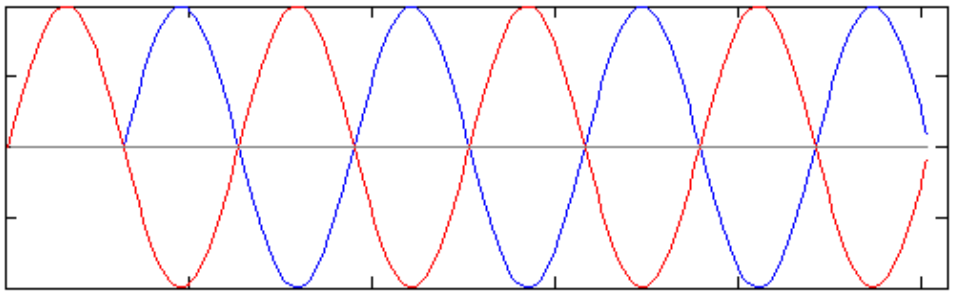
Figura 3.2: Oscilación de la carga ante un escalón.

Esta oscilación puede resultar peligrosa en determinados ambientes de trabajo sensibles o cuando la carga sea frágil por posibilidad de colisión con otros objetos.

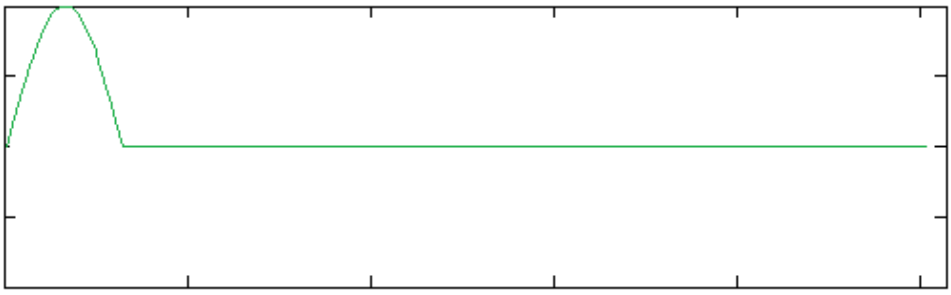
Smith[5], Calvert y Gimple [6] ya proponían una técnica sencilla para generar una respuesta no oscilatoria de un sistema ligeramente amortiguado a una entrada escalón. Esto se logra generando dos oscilaciones, de tal manera que una anule el efecto de la otra.

Pero es a partir de la publicación del artículo *"Preshaping command inputs to reduce system vibration"*, Singer y Seering [7], que esta técnica toma relevancia y aparece en múltiples artículos aplicada en naves espaciales, robots, grúas, etc.

Como un primer paso para entender cómo generar un comando que mueva un sistema sin vibraciones se comenzará por estudiar la reacción del sistema ante la entrada más simple. Al aplicar un impulso a un sistema, éste empezará a oscilar. Sin embargo, si se aplica un segundo impulso en el momento apropiado y con una amplitud determinada, la oscilación provocada por el primer impulso se anula (figura 3.3).



(a)



(b)

Figura 3.3: (a) Dos senoidales superpuestas. (b) Resultado de la suma.

Para conseguir eliminar la oscilación es necesario conocer la amplitud y el momento en el que se debe aplicar los impulsos. Si se conoce la frecuencia natural del sistema (ω) y el factor de amortiguación (ς), la vibración residual (V) que resulta de la aplicación de una secuencia de impulsos se puede describir como:

$$V(\omega,\varsigma) = e^{-\varsigma\omega t_n}\sqrt{C(\omega,\varsigma)^2 + S(\omega,\varsigma)^2} \quad (1)$$

donde,

$$C(\omega,\varsigma) = \sum_{i=1}^{n} a_i e^{\varsigma\omega t_i}\cos(\omega_d t_i) \quad (2)$$

$$S(\omega,\varsigma) = \sum_{i=1}^{n} a_i e^{\varsigma\omega t_i}\sin(\omega_d t_i) \quad (3)$$

Se define a_i como la amplitud de los impulsos, t_i como el tiempo en el que se deben aplicar los mismos, n es el número de impulsos en la secuencia y $\omega_d = \omega\sqrt{1-\varsigma^2}$.

En realidad, la ecuación 1 representa un porcentaje de vibración residual, es decir, indica la vibración causada por una secuencia de impulsos en comparación con la causada por un único impulso unitario. Si se establece que $V(\omega,\varsigma) = 0$ en la ecuación 1, se puede despejar las amplitudes y los tiempos de la secuencia de impulsos con los que se obtendría una vibración residual nula. Sin embargo, se debe imponer más restricciones para limitar el número de impulsos. En primer lugar, se establece que:

$$\sum a_i = 1 \quad (4)$$

Aun así, se puede satisfacer la condición con una larga combinación de impulsos de valores positivos y negativos. Por ello, para obtener una solución acotada, se establece que todos los impulsos deben tener una amplitud positiva.

$$a_i > 0, i = 1, 2, \dots, n \quad (5)$$

Por lo tanto, el problema a resolver consiste en encontrar una secuencia de impulsos que haga la ecuación 1 igual a cero y satisfaga las ecuaciones 4 y 5. Si se limita el número de impulsos a dos, el problema tiene cuatro incógnitas: las amplitudes de los impulsos (a_1, a_2) y el tiempo en el que se debe aplicar (t_1, t_2). Fijando $t_1 = 0$, el problema se reduce a tres incógnitas (a_1, a_2, t_2).

Para que la ecuación 1 sea cero, las ecuaciones 2 y 3 deben serlo también, ya que están elevadas al cuadrado en la ecuación 1. Por lo tanto:

$$0 = a_1 + a_2 e^{\varsigma\omega t_2} \cos(\omega_d t_2) \quad (6)$$

$$0 = a_2 e^{\varsigma\omega t_2} \sin(\omega_d t_2) \quad (7)$$

La ecuación 7 se satisface cuando $\sin(\omega_d t_2) = 0$. Esto ocurre cuando:

$$\omega_d t_2 = n\pi \Rightarrow t_2 = \frac{n\pi}{\omega_2} = \frac{nT_d}{2}, \; n = 1,2, \dots \quad (8)$$

Donde T_d es el periodo de vibración amortiguado. Este resultado indica que hay un número infinito de valores posibles para t_2, que se producen en múltiplos de la mitad del periodo de oscilación. Para cancelar la oscilación en el menor tiempo posible, se elije el menor valor para t_2:

$$t_2 = \frac{T_d}{2} \quad (9)$$

Además, fijando el número de impulsos a dos, se obtiene de la ecuación 4:

$$a_1 + a_2 = 1 \quad (10)$$

Con las ecuaciones 9 y 10 se puede definir, de forma sencilla, una secuencia de impulsos que anule la vibración del sistema. Pero un sistema real

no se puede mover mediante impulsos, por lo que es necesario transferir las propiedades de la secuencia de impulsos a una instrucción útil para el sistema.

Para ello se convoluciona la secuencia de impulsos con cualquier señal de control deseada. El resultado de esta convolución será el comando que ponga en movimiento el sistema. Si la secuencia de impulsos no causa ninguna vibración, el resultado de la convolución tampoco lo hará.

El proceso para generar este comando de entrada es lo que se conoce como *"Input Shaping"*. Se puede ver su interpretación gráfica en la figura 3.4.

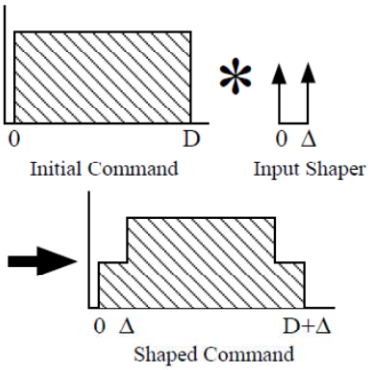


Figura 3.4: Proceso *"Input Shaping"*.

3.2. Implementación del algoritmo *"Input Shaping"*

El movimiento de una grúa es, por lo general, controlado por un operador humano. Éste activa el motor del carro durante un tiempo finito, de tal modo que el carro recorrerá una distancia determinada para luego pararse. La carga, en cambio, se mantendrá oscilando en torno a la posición del carro. Esto es porque la carga actúa como si de un péndulo se tratase.

En este proyecto en concreto se quiere estudiar la idoneidad del simulador Marilou para analizar los efectos sobre la carga de una grúa de algoritmos de control que reduzcan la oscilación. Por ello, se ha decidido desarrollar dos casos sencillos de *"Input Shaping"* aunque no sean los de mayor robustez.

De forma intuitiva, se han generado diversas *"Input Shaper"* buscando las que más reduzcan la vibración residual del sistema (V).

En primer lugar, teniendo en cuenta lo visto en el apartado anterior, se ha considerado que todos los impulsos se apliquen a la mitad del periodo de oscilación del sistema, ya que de esta manera se cumple que (3) es cero.

$$\sin(\omega \cdot t_i) = 0, \quad t_i = \frac{n \cdot T_d}{2}, n = 0,1,2, \ldots \tag{11}$$

Teniendo el tiempo en el que se aplican los impulsos, falta buscar una combinación de amplitudes (a_n) que hagan V (ecuación 1) lo más pequeño posible. Se ha calculado V para las diversas combinaciones. El resultado se muestra en la tabla 1:

Número de impulsos n	**Amplitud de impulsos** $[\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots]$	**Vibración residual** $\boldsymbol{V}$
2	[0.5, 0.5]	0.0727
3	[0.25, 0.5, 0.25]	0.0053
3	[0.2, 0.6, 0.2]	0.1663
3	[1/3, 1/3, 1/3]	0.2919
4	[0.2, 0.3, 0.3, 0.2]	0.0359
5	[0.1, 0.2, 0.4, 0.2, 0.1]	0.1497

Tabla 2: Tabla de Vibración residual para distintos *"Input Shaper"*.

Para todos los casos se ha despreciado el factor de amortiguación en la carga y se ha considerado la frecuencia angular del modelo linealizado

$$\omega = \sqrt{g/l} \tag{12}$$

donde *l* es la longitud del cable del que cuelga la carga y *g* la gravedad que afecta al sistema. Por lo tanto el periodo de oscilación es:

$$T_d = T = \frac{2\pi}{\omega} = 2\pi\sqrt{l/g} \tag{13}$$

Atendiendo a los resultados expuestos en la tabla 1 se ha decidido implementar el algoritmo con dos *"Input Shaper"* diferentes:

Opción 1: Control con dos impulsos.

No es la combinación que ofrece la menor vibración residual según los resultados teóricos obtenidos, pero sí la que reduciría la vibración en el menor tiempo posible ya que es la que menos impulsos debe aplicar.

Por lo tanto, esta secuencia de impulsos queda definida por:

$$a = a_1 = a_2 = 0.5 \qquad t_1 = 0 \qquad t_2 = \pi\sqrt{l/g}$$

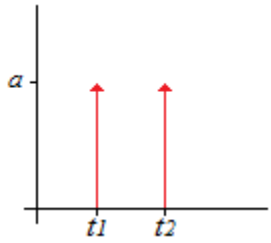


Figura 3.5: Secuencia de impulsos opción 1.

Opción 2: Control con tres impulsos.

De las secuencias estudiadas, se ha elegido la que ha obtenido la menor vibración residual teórica. Esta secuencia de impulsos está definida por:

$$a_1 = 0.25 \qquad a_2 = 0.5 \qquad a_3 = 0.25$$

$$t_1 = 0 \qquad t_2 = \pi\sqrt{l/g} \qquad t_3 = 2\pi\sqrt{l/g}$$

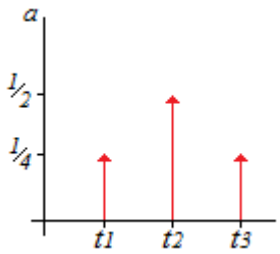


Figura 3.6: Secuencia de impulsos opción 2.

Este tipo de secuencias, como se ha visto en el apartado anterior, no se puede aplicar directamente a un sistema real. Se debe buscar qué comando genera el movimiento deseado en la carga.

En el algoritmo diseñado para aplicar esta secuencia al modelo, en vez de crear un escalón y luego aplicar la convolución como se vio en el apartado anterior, se generará directamente la secuencia de control del sistema.

Con los parámetros que se tienen, de haber seguido los mismos pasos que se definían en el apartado anterior, se crearía en primer lugar una entrada escalón con la velocidad deseada y se convolucionaría para obtener la señal final, representada en la figura 3.7.

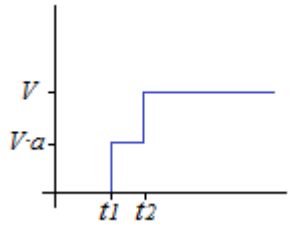


Figura 3.7: Señal para un movimiento sin oscilación con dos impulsos.

En vez de eso, conociendo la forma y los valores de la señal de velocidad deseada, se le irán dando las instrucciones al modelo para que la entrada resulte como la señal del *"Input Shaping"*.

La longitud de la cuerda, *l*, se obtiene mediante el sensor por ultrasonido. El sensor mide la distancia al suelo y, conociendo la altura a la que se encuentra el carro, se calcula la longitud del cable. Se toma la gravedad de la tierra como la gravedad que afecta al sistema. Con las clases definidas *Velocidad1* y *Velocidad2*, vistas en el capítulo anterior, se envían las consignas de velocidad. Esta función fija la velocidad del motor hasta que se le dé una nueva instrucción. Por otro lado, con la clase *Datos* se obtienen las vibraciones de la carga en todos los ejes, así como el tiempo de ejecución.

Gracias a estas funciones creadas en Marilou, se tiene un control completo de todos los parámetros necesarios para construir la señal *"Input Shaping"* y enviarla al modelo.

La señal se generará en tiempo real, asignando directamente al modelo los valores que conforman el escalón que resulta de cada una de las opciones propuestas. No se generará previamente, sino que con las lecturas del tiempo de ejecución se determinará cuándo se debe enviar cada señal.

CAPÍTULO 4

INTERFAZ GRÁFICA DE CONTROL Y VISUALIZACIÓN. MATLAB.

Para el control del modelo y la visualización de los resultados se ha creado una interfaz gráfica, GUI (del inglés *"Graphical User Interface"*), en MATLAB.

Una GUI es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

4.1. Interfaz gráfica de visualización y control.

El uso de GUI-s en MATLAB proporciona un entorno visual sencillo para interactuar con el modelo, mezclado con el poder computacional de MATLAB.

Se ha diseñado una interfaz que proporcione las herramientas necesarias para manejar el modelo y que simule los controles habituales de una grúa. Esta interfaz está desarrollada en el archivo ejecutable de Matlab *"RobotInterface.m"* (Anexo 1).

Este archivo contiene todas las funciones desarrolladas para este proyecto en Matlab. La función principal es *"RobotInterface"* que, en función de los parámetros de entrada que reciba, se encarga de llamar al resto de funciones.

Cuando se llama a esta función en Matlab, sin pasarle ningún parámetro de entrada, se ejecuta la función *"Initialisation3"*, la cual crea y despliega la interfaz gráfica diseñada para el control del modelo.

El resto de funciones contenidas en el archivo *"RobotInterface.m"* se ejecuta como resultado de accionar alguno de los botones contenidos en la interfaz. Estas funciones se verán al explicar el cometido de cada uno de los botones.

En este caso se han dispuesto los botones necesarios para conectar y desconectar el modelo, y para mover la grúa con o sin control activado, así como

casillas para indicar la velocidad de los motores o la posición deseada (Figura 4.1).

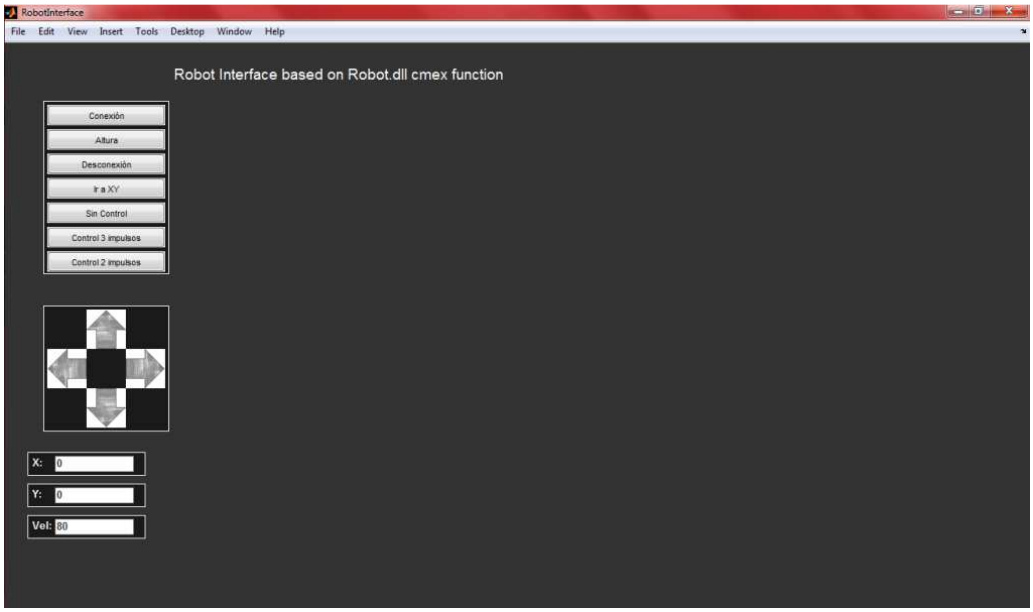


Figura 4.1: Interfaz para control del modelo.

En la parte superior izquierda se han dispuesto seis botones de acción:

- Conexión.
- Altura.
- Desconexión.
- Ir a XY.
- Sin Control.
- Control 3 impulsos.
- Control 2 impulsos.

A continuación están los botones de movimiento. Los botones arriba y abajo mueven los carros longitudinales y los botones izquierda y derecha mueven el carro transversal.

En el lado inferior izquierdo se encuentran las casillas de obtención de datos. En las dos primeras, X e Y, se puede rellenar con las coordenadas de la

posición deseada en metros respecto al centro de la grúa (origen de las coordenadas). En el último casillero, Vel, se indica la velocidad en grados por segundo a la que deben ponerse los motores.

La zona central y derecha se reserva para la visualización del modelo en funcionamiento y para la representación de las gráficas de resultados (figura 4.2).

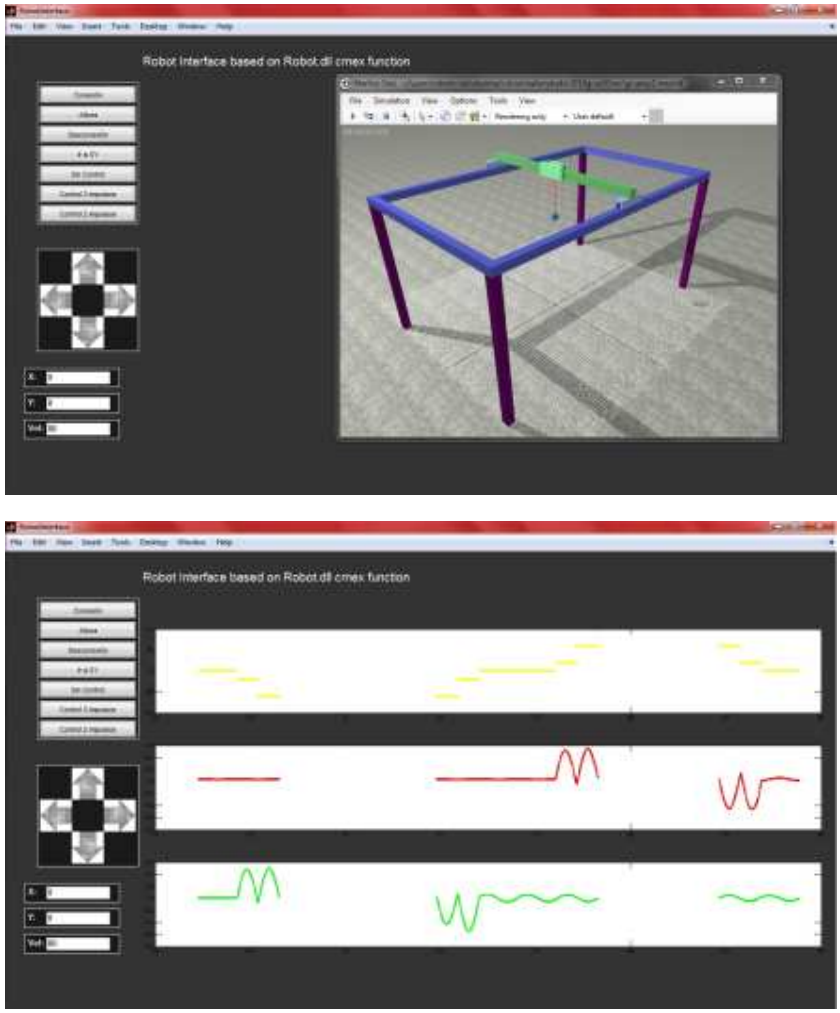


Figura 4.2: Visualización de la simulación del modelo y de las gráficas.

4.2. Botones de acción y movimiento. Funciones desarrolladas en Matlab.

Cada uno de los botones vistos provoca interacciones diferentes con el modelo. Algunos llaman a una función para que interactúe con el modelo, otros fijan parámetros que serán importantes a la hora de poner la grúa en movimiento.

A continuación se detalla las acciones que hay detrás de la pulsación de cada uno de los botones.

4.2.1. Conexión.

Este botón llama a la función creada en MATLAB *"Connexion"*.

Esta función se encarga, en primer lugar, de inicializar todas las variables que van a ser necesarias y almacenarlas en dos archivos de datos, *"data.mat"* y *"signo.mat"*, para que puedan ser utilizadas siempre que sea necesario.

- ***"data.mat"*** registra los datos de velocidad, oscilación y tiempo para que puedan ser representados al final de la prueba. La variable "T" es el tiempo de ejecución, "C" se corresponde con la oscilación en los tres ejes, "x" almacena el valor equivalente a la mitad del periodo de oscilación, "y" guarda la velocidad del motor en cada momento y "j" es el número de registros guardados.

- ***"signo.mat"*** almacena el valor de unas banderas ("*flags*") necesarias para el funcionamiento. La bandera "b" indica si la velocidad que se debe aplicar a los motores es positiva o negativa, "onoff" refleja si el motor ya está en marcha o si está parado, "control" dice si el movimiento se debe hacer sin aplicar el algoritmo, aplicando el de dos impulsos o el de tres y, por último, "XoY" es para saber si se debe mover el carro transversal o los longitudinales.

Estos archivos se cargarán cada vez que se vaya a realizar alguna acción que implique la modificación o representación de alguno de los parámetros que almacenan. Asimismo, después de su uso se guardarán para que puedan ser usados en otras funciones.

Tras la inicialización, la función realiza la conexión con el modelo creado en Marilou. Para ello accede al archivo *"datos.m"*, que debe contener en la primera línea la IP del modelo y en la segunda, la ruta para acceder al mismo. Estos datos son necesarios para llevar a cabo la conexión. El archivo *"datos.m"* utilizado en este proyecto es como el representado en la figura 4.3.

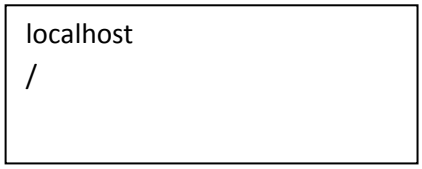


Figura 4.3: Ejemplo del contenido del archivo *"datos.m"*

Para completar la conexión de la ejecución en Matlab con la simulación del modelo en Marilou, se realiza la llamada a los programas creados en C++ (Capítulo 2) con la bandera correspondiente 'c' (Véase Tabla 1, Capítulo 2).

TestConnexion = Robot ('c', IP, ROBOT);

4.2.2. Altura.

Este botón llama a la función *"longitud"* del programa desarrollado en Matlab para obtener la longitud del cable del que cuelga la carga y calcular el periodo de oscilación. Se debe ejecutar cada vez que se conecte al modelo y antes de realizar ninguna otra acción.

Para calcular la longitud del cable, se recoge la medida facilitada por el sensor de distancia mediante ultrasonido ubicado en la carga del modelo y se calcula la diferencia con la altura desde la que cuelga el cable. La medida del ultrasonido se obtiene mediante la función:

[~,l]=Robot('h',l);

Teniendo la longitud del cable se puede sustituir en $\omega = \sqrt{g/l}$ y obtener la frecuencia de oscilación y, con ésta, el periodo.

La mitad del periodo se guarda como "x" en el archivo *"data.mat"* que se vio en el apartado "Conexión" para que se pueda acceder a ella cuando sea necesario. Este archivo se elimina y vuelve a crear con cada nueva conexión, por eso es necesario ejecutar la función *"longitud"* cada vez que se conecta con el modelo.

4.2.3. Desconexión.

Este botón llama a la función *"Disconnexion"* creada en Matlab. Ésta carga los datos recogidos durante la manipulación del modelo (*"data.mat"*) y representa la velocidad y la oscilación en los ejes X e Y respecto al tiempo. Estas gráficas aparecerán en la parte libre de la interfaz, como ya se indicó en el apartado anterior.

Gracias a esta representación se puede apreciar el efecto que causa el algoritmo diseñado en el modelo.

Además, esta función manda la orden de detener los motores que aún estén en marcha y desconecta el programa realizado en Matlab de la simulación del modelo en Marilou. Para llevar a cabo estas acciones, se realizan llamadas similares a la vista en la función *"Connexion"* pero con diferentes parámetros.

Robot ('V',0.0);

Robot ('v',0.0);

Robot ('d');

4.2.4. Ir a XY

Este botón llama a la función *"iraXY"* del programa *"RobotInterface.m"* creado en Matlab. Es una de las funciones más complejas que se ha desarrollado, ya que debe calcular el movimiento a realizar en función de la posición a la que desea llegar. Se desarrolla únicamente con el control de tres impulsos, es decir, se le aplica el *"Input Shaper"* de la opción 2 del capítulo 2 (Figura 4.4) con una velocidad de rotación para todos los motores prefijada a 80 grados por segundo.

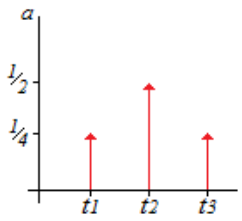


Figura 4.4: *"Input Shaper"* de tres impulsos.

De esta manera, si se define la velocidad que se desea alcanzar Amax=80, se debe aplicar una velocidad Amax·1/4=20 durante el primer medio periodo, 3/4·Amax=60 durante el segundo medio periodo y Amax=80 a partir de ahí. Se mantiene v3 hasta llegar a una cierta distancia de la posición deseada, momento en el cual se aplica la secuencia inversa hasta llegar a una velocidad de 0.

Lo primero que realiza la función *"iraXY"* cuando se ejecuta es leer la posición deseada en metros, que se debe haber introducido previamente en las casillas X e Y de la interfaz gráfica. Conociendo el radio de las ruedas de los carros del modelo y el número de medidas por vuelta que tiene el encoder, se calcula la cuenta que debe tener el encoder para estar en la posición indicada.

Por otro lado, solicita al modelo la medida del encoder de cada motor (*'Robot('e',0.0);'*). Comparando estas medidas se evalúa, para cada eje por separado, si se debe realizar algún movimiento y en qué sentido. Si la posición deseada es diferente de la posición actual de la carga, se llamará a las funciones *"iraX"* o *"iraY"* según corresponda. El eje X corresponde a un movimiento del carro transversal y el eje Y, a los carros longitudinales.

Las funciones *"iraX"* e *"iraY"* se diferencian únicamente en que una se encarga del movimiento del carro transversal y ejecuta *'Robot('V', velocidad);'* y la otra actúa sobre los carros longitudinales con la función de control sobre el modelo de Marilou *'Robot('v', velocidad);'*.

En primer lugar, como se van a modificar las consignas de velocidad y se van a registrar nuevas oscilaciones, se carga el archivo *"data.mat"* para añadir los nuevos datos recogidos.

Durante los primeros dos segundos después de la llamada se mantiene el estado del sistema y se toman medidas para poder comparar con las variaciones que se producirán al comenzar con el movimiento.

Después de esto comienza el movimiento de la carga con una velocidad correspondiente a un cuarto de la velocidad a alcanzar:

*A=Amax*1/4*b;*

La variable "b" es la encargada de marcar el sentido del movimiento del carro que se ha evaluado en la función anterior, *"iraXY"*. Será b=1 para asignar una velocidad positiva y b= -1 para la negativa.

Se mantendrá esta consigna de velocidad hasta que haya pasado un tiempo equivalente a la mitad del periodo de oscilación o hasta que se detecte que se ha llegado a la posición deseada.

Una vez pasado el periodo de tiempo indicado, se cambia la consigna de velocidad a:

*A=3/4*Amax*b;*

Cuando haya pasado otra segunda mitad de periodo se vuelve a cambiar la velocidad deseada para fijarla como:

*A=Amax*b;*

Con el motor a la velocidad deseada se evalúa el valor del encoder para ver cuándo se aproxima la carga a la posición deseada. Cuando se encuentra a la distancia de frenado correspondiente para la velocidad prefijada en esta función, se comienza el mismo proceso de puesta en marcha pero a la inversa.

Se fija una velocidad durante el primer medio periodo *'A=3/4*Amax*b;'*, durante el segundo medio periodo *'A=1/4*Amax*b;'* y, finalmente, se frena el motor asignándole una velocidad *'A=0;'*.

En todo momento y hasta dos segundos después de frenado el motor se van tomando medidas de oscilación que se almacenan en la variable "C" del archivo *"data.mat"*. Esto se hace para poder, más tarde, representar la oscilación provocada por los movimientos inducidos en la carga.

Todo este proceso se realiza primero para el eje X, función *"iraX"*, buscando la posición correspondiente en este eje y después, para el eje Y, función *"iraY"*.

4.2.5. Sin Control, Control 3 impulsos y Control 2 impulsos.

Estos tres botones dan valor a la variable "control" que se almacena en el archivo *"signo.mat"*. Se dará un valor 0 a la variable cuando se seleccione la opción "Sin Control", 1 para el caso de "Control 3 impulsos" y 2 si se pulsa "Control 2 impulsos".

Se han desarrollado funciones diferentes para cada una de estas tres opciones. Esta variable se usará para decidir a qué función se debe llamar cuando se pulsen los botones de movimiento en función de la opción seleccionada.

4.2.6. Botones de movimiento.

Estos botones tienen un funcionamiento característico, cuando se pulsa por primera vez comienza el movimiento de la grúa en el sentido indicado. Para pararlo, y antes de realizar ninguna otra acción, se debe volver a pulsar el mismo botón.

Al pulsar cualquiera de estos botones se ejecuta la función *"QueHacer"*. Ésta evaluará si la grúa está o no en movimiento (variable "onoff") y, sabiendo el valor de la variable "control", determinará la acción a llevar a cabo. Hay seis funciones posibles en función de las opciones que se hayan seleccionado y del estado del modelo.

4.2.6.1. Funciones *"OnNoControl"* y *"OffNoControl"*:

Si la opción seleccionada era "Sin Control" la función *"QueHacer"* llamará a alguna de estas dos funciones. Si la grúa se encuentra parada se seleccionará la función *"OnNoControl"*, que se encarga de poner en movimiento la carga sin aplicar ningún tipo de algoritmo de control. Si, por el contrario, la grúa ya estaba en movimiento, se llamará a *"OffNoControl"* para detenerla.

Igual que se hace en las funciones *"IraX"* e *"IraY"*, desde dos segundos antes de mandar la orden al modelo hasta dos segundos después, se recogerán todos los datos de oscilación de la carga en la variable "C".

Para decidir en qué eje y en qué sentido se debe realizar el movimiento se usan las variables "b" y "XoY" del archivo *"signo.mat"*. El valor de estas variables se establece al pulsar los botones de movimiento.

En esta ocasión se pasará al modelo la consigna definitiva de velocidad directamente (figura 4.5).

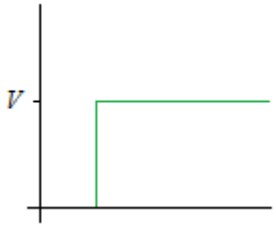


Figura 4.5: Señal de Velocidad sin aplicar algoritmo.

4.2.6.2. Funciones *"Steps3"* y *"Stops3"*:

Se usan estas funciones cuando se selecciona el control con tres impulsos. La función *"Steps3"* se encargará de iniciar el movimiento de la grúa y *"Stops3"* de detenerlo.

Se utiliza la misma secuencia *"Input Shaper"* vista en la función *"IraXY"*, [0.25 0.5 0.25].

En la función *"Steps3"* se aplicará una primera consigna de velocidad *'A=Amax*1/4*b;'* transcurridos dos segundos desde que se da la orden (la variable "b" determina el sentido del movimiento). Cuando haya pasado el tiempo equivalente a medio periodo de oscilación, se aplicará al modelo el segundo valor de velocidad *'A=Amax*3/4*b;'*. Finalmente, transcurrido otro semiperiodo, se asigna la consigna de velocidad final deseada *'A=Amax*b;'*. En la función *"Stops3"* se aplica la misma secuencia en orden inverso, *'A=Amax*3/4*b;'* en primer lugar, a continuación *'A=Amax*1/4*b;'* y para terminar y detener el movimiento de la grúa, se manda una consigna de velocidad *'A=0;'*.

Desde el momento en que se presiona el botón hasta dos segundos después de terminar de aplicar el *"Input Shaping"*, se toman medidas de oscilación y de tiempo de ejecución gracias al giroscopio instalado en el modelo realizado en Marilou. Estas medidas se almacenan hasta que se pulsa el botón de "Desconexión", momento en el cual se representan para poder evaluar la efectividad del algoritmo y del modelo.

4.2.6.3. Funciones *"Steps2"* y *"Stops2"*:

Estas son las funciones que se encargan de realizar el movimiento de la grúa cuando se selecciona un control con dos impulsos. Las funciones son muy

similares a las dos anteriores (*"Steps3"* y *"Stops3"*) pero aplican un *"Input Shaper"* de dos impulsos.

En este caso, y como se explicó en el capítulo 3, se ha usado un *"Input Shaper"* de dos impulsos en el que ambos tienen el mismo valor, [0.5 0.5]. Eso se traduce en un *"Input Shaping"* equivalente a aplicar dos escalones de igual valor.

Así, la función *"Steps2"* enviará en primer lugar una consigna de velocidad *'A=Amax*1/2*b;'* y, medio periodo después, aplicará la velocidad indicada *'A=Amax;'* para iniciar un movimiento de la carga con la menor oscilación posible. La función *"Stops2"* aplicará la misma secuencia pero su segunda orden será *'A=0;'* para detener el movimiento de la grúa.

Igual que ocurría con las otras funciones, se estará recogiendo valores de oscilación de la carga desde dos segundos antes de comenzar a aplicar el *"Input Shaping"* hasta dos segundos después.

4.3. Guía de uso de la interfaz.

A continuación se describe los pasos a seguir para un correcto funcionamiento de la interfaz y su conexión con el modelo.

Paso 0. Antes que nada, es necesario ejecutar la simulación de Marilou y la interfaz de Matlab.

Para desplegar la interfaz, basta con introducir en Matlab por línea de comandos el comando:

```
>> RobotInterface;
```

Es necesario que la carpeta de ejecución de Matlab corresponda con la carpeta en la que se encuentra el archivo.

Para comenzar la simulación de Marilou se debe ejecutar el modelo correspondiente sobre el que se quiera actuar (cuerda1 o cuerda2), tal y como se explicó en el apartado 2.6 del Capítulo 2 ("Simulación del modelo").

Paso 1. Antes de empezar a usar la interfaz se debe tener el modelo de Marilou activo, ya que si no Matlab no tendrá ningún modelo que conectar. Esto quiere decir que la simulación del modelo en Marilou debe estar activa como se ve en la figura 4.6, el botón de *"play"* pulsado y el tiempo de ejecución corriendo.

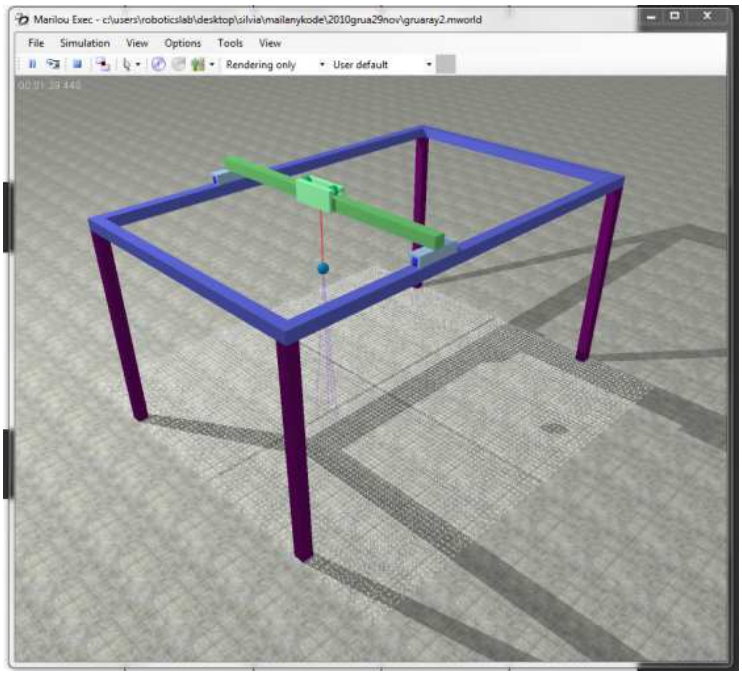


Figura 4.6: Simulación del modelo de Marilou en marcha.

Paso 2. Se pulsan los botones "Conexión" y "Altura" para conectar el programa de Matlab a la ejecución de Marilou y obtener la longitud del cable. Además, sería recomendable revisar el parámetro de velocidad por defecto (fijado a 80 grados por segundo) por si se quiere cambiar.

Paso 3. En este momento se tiene diversas opciones. Se debe elegir si se quiere usar los botones de movimiento para controlar la grúa (rama A) o si se desea fijar una posición de destino (rama B).

Paso 4 – Rama A. Se debe seleccionar el modo de funcionamiento: Sin control, control de 3 o 2 impulsos, que se desea usar.

Paso 5 – Rama A. Usando los botones de movimiento se traslada la carga al punto deseado. Hay que tener en cuenta que se debe pulsar el botón una primera vez para iniciar el movimiento y antes de cambiar de dirección, una segunda vez para pararlo. La grúa solo puede mover un eje cada vez.

Paso 4 – Rama B. Si se quiere usar el modo por coordenadas, en primer lugar se debe introducir las coordenadas de destino deseadas en las casillas "X" e "Y".

Paso 5 – Rama B. Pulsar el botón "Ir a XY" para que la carga se traslade al punto indicado.

Se puede combinar cualquiera de estas opciones en una misma ejecución. Por ejemplo, se puede usar los botones de movimiento para trasladar la carga a un punto cualquiera y después usar el modo por coordenadas para volver a la posición inicial (0,0).

Aunque se puedan combinar, no es aconsejable usar el modo sin control y después alguno con control ya que podrían no eliminarse las vibraciones creadas por el primero.

Paso 6. Para finalizar la conexión establecida con el modelo de Marilou y representar las gráficas de la oscilación que se ha registrado en la carga durante el tiempo de ejecución, se debe pulsar el botón “Desconexión”.

CAPÍTULO 5

RESULTADOS OBTENIDOS

Como se explicó en el Capítulo 2, debido a las limitaciones de Marilou, se han creado dos modelos diferentes en función de la forma de sostener la carga. Por un lado se tiene el modelo de cuerda rígida (cuerda 1), donde la cuerda está modelada por un elemento rígido y sin masa; y por otro, el modelo de cuerda por cápsulas (cuerda 2), en el que se modela la cuerda con un conjunto de cápsulas unidas entre sí.

La respuesta de estos dos modelos ante una entrada escalón es diferente en cada caso. En ambos, como se ha visto que debe suceder ante una perturbación de este tipo, se genera una oscilación en la carga cuya frecuencia de oscilación se corresponde aproximadamente con $\omega = \sqrt{g/l}$. Pero además, en el caso de la cuerda 2 (cuerda mediante cápsulas), se detecta una vibración a una frecuencia mayor.

En la figura 5.1 se representan las oscilaciones registradas en un modelo con un tipo de cuerda 1 (cuerda rígida) tanto para el eje X (rojo) como para el eje Y (verde) ante un escalón de velocidad aplicado sobre el eje Y (azul).

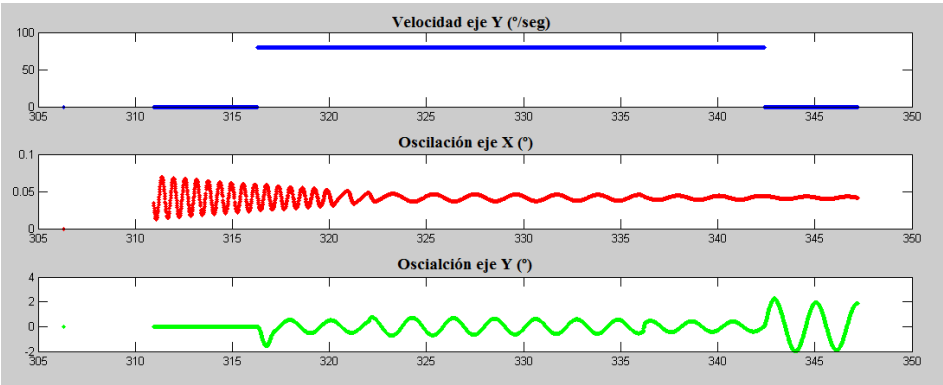


Figura 5.1: Respuesta ante escalón del modelo Cuerda 1.

La respuesta de la carga es la esperada según lo visto en el capítulo 3. Ante la perturbación recibida, el sistema reacciona generando una oscilación en el mismo eje como si de un péndulo se tratase. Se observa, además, una

oscilación en el eje X. Esta es despreciable y se debe a los ajustes del modelo al comienzo de la ejecución.

Si se observa ahora en la figura 5.2 la respuesta del modelo realizado con el tipo de cuerda 2 (cápsulas) se aprecia una variación notable.

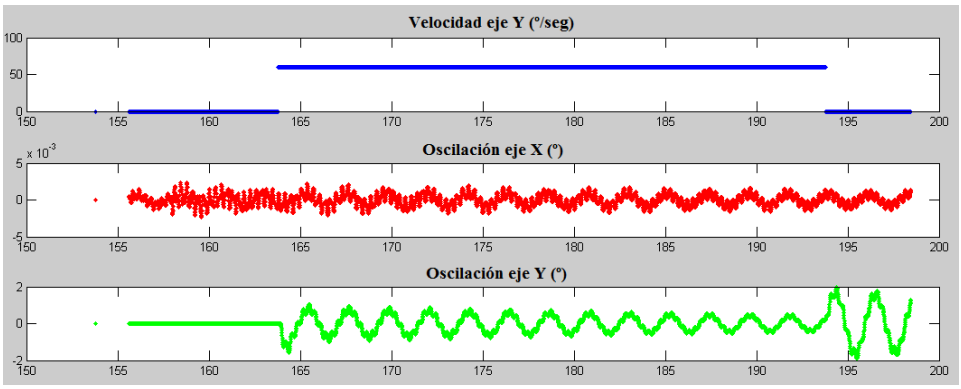


Figura 5.2: Respuesta ante escalón del modelo Cuerda 2.

En este caso, además de la oscilación esperada en el eje Y, se detecta una oscilación a una frecuencia mayor. Esta frecuencia corresponde a la de un péndulo de longitud igual a la del tamaño de una cápsula. Al estar la cuerda modelada por pequeñas cápsulas, se introduce una segunda perturbación sobre el modelo, la carga oscila a dos frecuencias diferentes.

Para analizar la idoneidad de ambos modelos para un estudio de oscilación sobre la carga de una grúa, se ha aplicado a cada uno de ellos los dos *"Input Shaping"* desarrollados en este proyecto.

5.1. *"Input Shaping"* de dos impulsos.

La señal de *"Input Shaper"* de dos impulsos seleccionada es la que corresponde con los valores:

$$a = a_1 = a_2 = 0.5 \qquad t_1 = 0 \qquad t_2 = \pi\sqrt{l/g}$$

Se ha aplicado a ambos modelos el *"Input Shaping"* resultante de hacer la convolución de este *"Input Shaper"* con un escalón de velocidad de 80º/seg (figura 5.3).

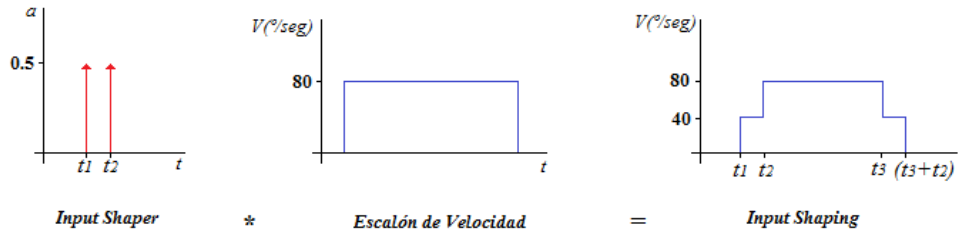


Figura 5.3: *"Input Shaping"* de dos impulsos.

Como se vio en el Capítulo 3, esta opción no elimina completamente la vibración sino que quedaría una vibración residual V=0.0727, con lo cual, lo esperado es que se reduzca la vibración observada ante la aplicación de un escalón de velocidad, pero sin llegar a eliminarse.

En la figura 5.4 se observa la respuesta del modelo con cuerda rígida ante este *"Input Shaping"*. Si se mira la representación de la oscilación en el eje Y (verde) se ve como ante la primera señal de velocidad comienza a oscilar con una amplitud similar a la vista en el caso sin control. Pero al hacer la segunda subida de velocidad, esta oscilación se reduce considerablemente, se queda en una oscilación con una amplitud aproximada de 0.1º.

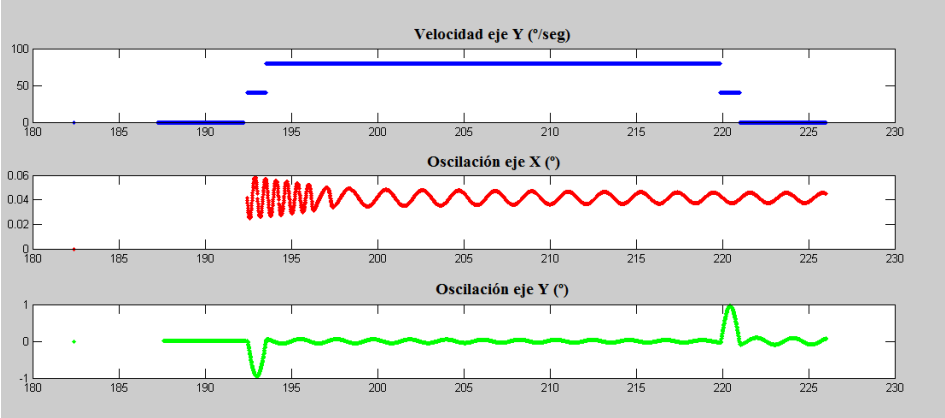


Figura 5.4: Respuesta del modelo de cuerda rígida ante un *"Input Shaping"* de dos impulsos. (NOTA: Se han suprimido los primeros segundos de oscilación en el eje X para una mejor visualización)

La figura 5.5 representa la respuesta del modelo de cuerda por cápsulas ante el *"Input Shaping"* de dos impulsos. En este caso, aunque se reduce la amplitud de la oscilación principal, la correspondiente a la longitud completa de la cuerda, se puede ver como la vibración provocada por las cápsulas se mantiene.

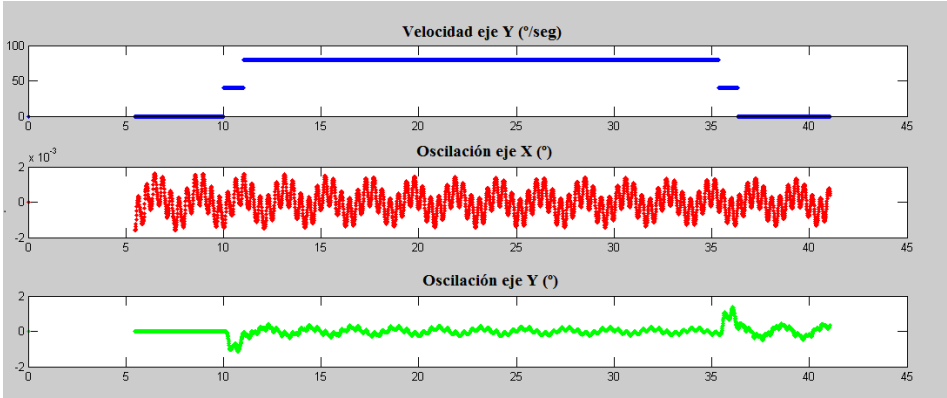


Figura 5.5: Respuesta del modelo de cuerda por cápsulas ante un *"Input Shaping"* de dos impulsos.

En ambos casos se observa que la acciones realizadas sobre el eje Y no afectan a las oscilaciones del eje X. Las vibraciones que se presentan en el eje X son resultado del ajuste de las piezas del modelo. Es decir, si la rueda del carro transversal quedó con algún milímetro de distancia respecto a la viga, al poner en marcha la simulación del modelo el carro se acoplará a la viga produciendo una pequeña vibración en la carga que cuelga de él. En el eje Y no se aprecia esta vibración porque la oscilación producida por el movimiento de la carga es muy superior.

5.2. *"Input Shaping"* de tres impulsos.

Como se ha indicado en los capítulos anteriores, el segundo *"Input Shaping"* que se ha probado es el que ha dado menor vibración residual de entre los que se probaron, V=0.0053. Los parámetros que definen el *"Input Shaper"* son:

$$a_1 = 0.25 \qquad a_2 = 0.5 \qquad a_3 = 0.25$$

$$t_1 = 0 \qquad t_2 = \pi\sqrt{l/g} \qquad t_3 = 2\pi\sqrt{l/g}$$

En esta ocasión, también se ha seleccionado una velocidad de 80º/seg para que se pueda comparar fácilmente entre ambos tipos de control (figura 5.6).

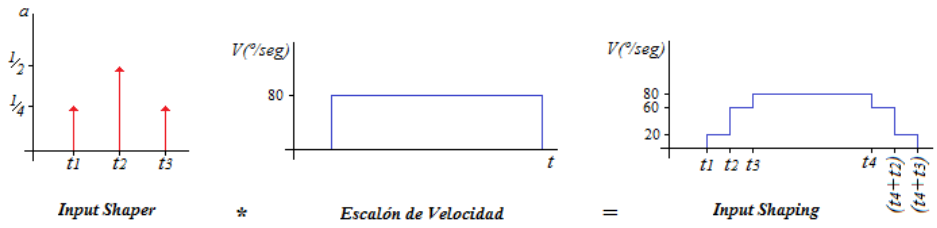


Figura 5.6: *"Input Shaping"* de tres impulsos.

Al ser la vibración residual para esta señal de *"Input Shaping"* menor que para el anterior, se espera que las oscilaciones se reduzcan más.

El la figura 5.7 se puede ver la respuesta del modelo con cuerda rígida a esta señal. Como ya se adelantaba, se aprecia que al aplicar este *"Input Shaping"* la oscilación es mucho menor. Únicamente se observan las dos ondas provocadas por los cambios de velocidad. Una vez se alcanza la velocidad deseada la oscilación a penas se aprecia. La vibración tiene una amplitud aproximada de 0.02º.

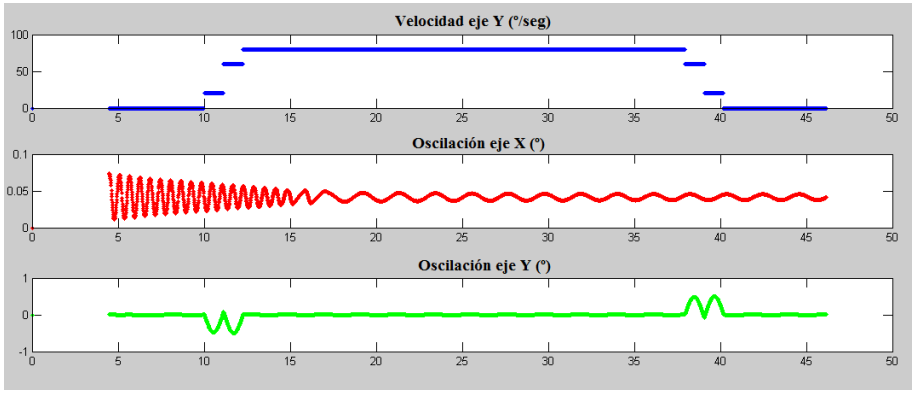


Figura 5.7: Respuesta del modelo de cuerda rígida ante un *"Input Shaping"* de tres impulsos.

En la figura 5.8 se representan las oscilaciones obtenidas ante la señal *"Input Shaping"* de tres impulsos para el modelo con cuerda por cápsulas. En este caso la oscilación principal se reduce hasta tal punto que no se aprecia en comparación con el efecto que causan las cápsulas.

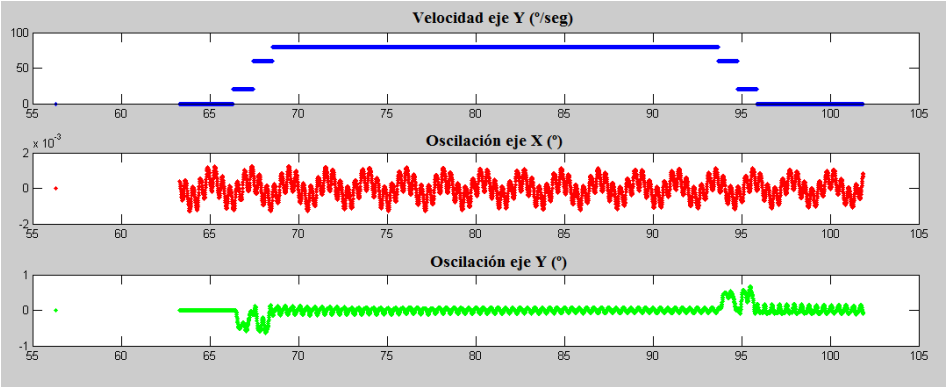


Figura 5.8: Respuesta del modelo de cuerda de cápsulas ante un *"Input Shaping"* de tres impulsos.

5.3. Movimientos encadenados.

Para terminar es necesario comprobar cómo actúa el sistema ante una consecución de comandos varios en ambos ejes, siempre aplicando señales *"Input Shaping"* de tres impulsos, ya que se ha comprobado que es el que mejor respuesta tiene de los dos que se han probado.

En la figura 5.9, se traslada la carga a un punto determinado usando los botones de movimiento de la interfaz y a continuación se vuelve a la posición de origen. Esto último se hace usando la función *"IraXY"* vista en el capítulo anterior.

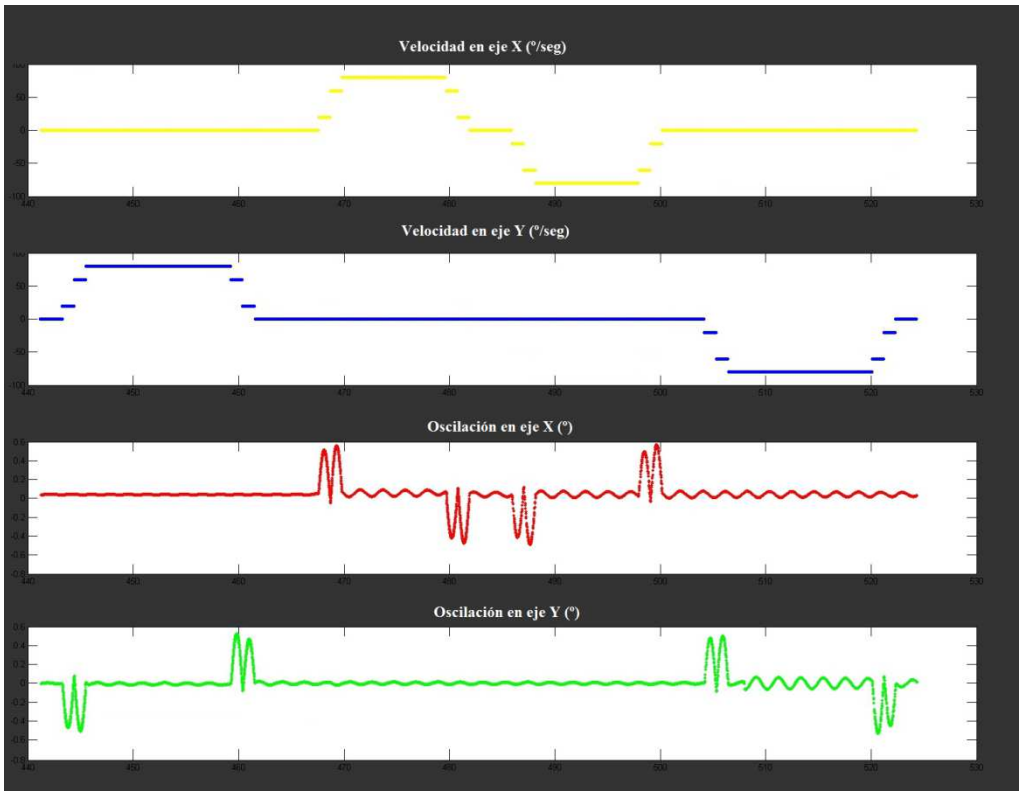


Figura 5.9: Respuesta del modelo de cuerda rígida ante una consecución de *"Input Shaping"*.

Se puede ver como los movimientos inducidos en un eje de acción no afectan al otro. Aun así, se debe recordar que antes de empezar el movimiento en un eje se debe haber detenido el del otro.

Tras las pruebas realizadas se puede decir que un algoritmo de control por *"Input Shaping"* sencillo sirve para reducir notablemente la oscilación que se produce al manipular el sistema. Además, se aprecia que, tal y como anticipaban los cálculos teóricos, el *"Input Shaping"* de tres impulsos es mejor que el de dos.

CAPÍTULO 6

CONCLUSIONES

El objetivo de este proyecto ha sido, desde un principio, evaluar la capacidad de la herramienta de modelado y simulación Marilou para simular y probar un algoritmo anti-balanceo en una grúa puente y su entorno de trabajo.

En el Capítulo 2 se ha visto que, pese a todas las herramientas que facilita el programa Marilou, no ha sido posible modelar correctamente un cable o cuerda que sostenga la carga. Esto es porque se trata de un programa enfocado a trabajar con elementos sólido-rígidos.

A pesar de esto se ha intentado modelar el cable que sostiene la carga usando las herramientas que facilita el programa. De esta manera, se han desarrollado dos modelos diferentes: una cuerda conformada por un número determinado de cápsulas unidas entre sí, como formando una cadena; y una modelada con un elemento rígido y sin masa que hace de unión entre la carga y el carro del que debería colgar.

Ninguna de estas dos opciones es del todo correcta.

El modelo realizado con un elemento rígido responde a una situación ideal en la que la cuerda no tiene flexibilidad ninguna y, por lo tanto, no influiría a la hora de aplicar el algoritmo.

Por otro lado, el modelo de la cuerda conformada por cápsulas introduce perturbaciones en la respuesta del sistema. Como se ha visto en el Capítulo 5, las cápsulas afectan a la respuesta del sistema con su propia frecuencia de oscilación, provocando perturbaciones. Quizás este modelo de cuerda sería una buena idea si se pudiera reducir considerablemente el tamaño de las cápsulas, de esta forma la cuerda quedaría conformada por un montón de pequeñas uniones que apenas distarían de la dinámica real de una cuerda. Pero la simulación falló cuando se intentó reducir el tamaño de las cápsulas y aumentar el número de las que conformaban la cuerda.

Por todo esto, aunque el programa Marilou ofrece múltiples herramientas para el modelado, simulación y diseño de robots, no es la herramienta idónea para trabajar en proyectos que incluyan elementos flexibles.

A pesar de esto, se ha podido comprobar cómo el algoritmo, incluso en sus desarrollos más sencillos, consigue disminuir notablemente las vibraciones en el sistema.

Para un futuro uso de este algoritmo en un sistema real, sería recomendable establecer unas especificaciones de oscilación máxima permitida y estudiar la posibilidad de desarrollar un algoritmo más robusto, como lo que proponen Singh y Singhose en su artículo *"Tutorial on Input Shaping"* [4], o desarrollar un sistema que elimine otras perturbaciones debidas a elementos externos que se puedan producir durante el movimiento de la carga, como el desarrollado por Garrido y Abderrahim [2].

BIBLIOGRAFÍA

[1] TechnoFusión. National Centre for Fusion Technologies. Scientific-Technical Report.

[2] Garrido, S., Abderrahim, M., Giménez, A., Díez, R. y Balaguer, C. 2008. "Anti-swinging Input Shaping Control of an Automatic Construction Crane" IEEE Transactions on Automation Science & Engineering. Vol. 5. No. 3. pp.549-557.

[3] D.Blanco; S.Garrido; L.Moreno. Simulation Platform for Anti-Swinging Control of Automatic Overhead Cranes. 1st Workshop on Fusion Technologies and the contribution of TECHNOFUSIÓN Editores: Angel Ibarra, José Manuel Perlado y Manuel Ferre ISBN (edición impresa): 978-84-7484-239-5 ISBN (edición digital): 978-84-7484-240-1 . Vol. 1. No. 1. pp.129-142. 2011.

[4] Singh, T., Singhose, W. "Tutorial on Input Shaping/Time Dealy Control of Maneuvering Flexible Structures" American Control Conference, Anchorage, AK, 2002, pp. 1717-1731.

[5] Smith, O. J. M., "Posicast Control of Damped Oscillatory Systems", Proc. of the IRE, 1957, pp 1249-1255.

[6] Calvert, J. F. and Gimpel, D. J., "Method and Apparatus for Control of System Output Response to System Input", U.S. Patent #2,801,351, 1957.

[7] Singer, N. C., and Seering, W. P., "Preshaping Command Inputs to Reduce System Vibrations", *ASME J. of Dynamic Systems, Measurement and Control*, Vol. 112, 1990, pp 76-82.

[8] Página web del programa Marilou "http://www.anykode.com/index.php"

[9] Página web de soporte a Matlab "http://www.mathworks.es"

[10] Página web del proyecto TechnoFusión "http://www.technofusion.es/"

[11] Manual de ayuda del programa Marilou.

[12] Manual de ayuda del programa Matlab.

ANEXOS

Anexo 1: RobotInterface.m
Anexo 2: cmex.cpp
Anexo 3: Robot.cpp
Anexo 4: Robot.h